AFRL-AFOSR-VA-TR-2016-0286

DDDAS-based Resilient Cyberspace (DRCS)

Salim Hariri
ARIZONA UNIV BOARD OF REGENTS TUCSON
888 N. EUCLID AVENUE
TUCSON, AZ 85722-3308

08/03/2016
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/RTA2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 07/31/2016	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 1 May 2012 - 30 April 2016

4. TITLE AND SUBTITLE
DDDAS-based Resilient Cyberspace (DRCS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-12-1-0241

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Hariri, Salim

5d. PROJECT NUMBER
FA9550-12-1-0241

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The University of Arizona
Tucson, AZ 85721

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
875 N. Randolph St., Rm. 3112, Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION A: Distribution approved for public release.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
It is critically important for Dynamic Data Driven Application Systems (DDDAS) to operate normally in spite of cyberattacks. In this project, we developed resilient algorithms and middleware to build resilient DDDAS (rDDDAS) that can tolerate cyberattacks, faults or accidents that might have been triggered by malicious or natural events. We evaluated the performance, overhead, and feasibility of our rDDDAS design methodology to develop resilient scientific and engineering applications.

15. SUBJECT TERMS
Dynamic Data-Driven Application Systems (DDDAS), resiliency, resilient DDDAS (rDDDAS), Moving Target Defense (MTD), Software Behavior Encryption (SBE), redundancy, diversity, shuffling, cloud computing, resilient cloud services

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Hariri, Salim
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	48	520-621-4378

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATE COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33315-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report. e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Final Report

DDDAS-based Resilient Cyberspace (DRCS)

Contract/Grant #: FA9550-12-1-0241

Reporting Period: 1 May 2012 to 30 April 2016

Principle Investigator: Salim Hariri, Ph.D

Department of Electrical and Computer Engineering

The University of Arizona

Submitted to

Dr. Frederica Darema, AFOSR\RSE

fredrica.darema@afosr.af.mil

1. Project Summary

It is expected that Dynamic Data-Driven Application Systems (DDDAS) applications will be widely deployed to optimize the operations of cyber infrastructures and mission critical applications. Consequently, it is critically important for DDDAS environments to operate resiliently against any type of cyberattacks (either known or unknown). In this project, we focused on the development of resilient algorithms, middleware, and DDDAS-based applications that can continue to operate normally in spite of the occurrence of cyberattacks, faults or accidents that could be trigged by malicious or natural events. The main contributions of this research are the followings:

1) **A methodology to build resilient DDDAS (rDDDAS) environment:** rDDDAS utilizes Moving Target Defense (MTD) and Software Behavior Encryption (SBE) techniques to make it extremely difficult for hackers to exploit existing vulnerabilities or compromise DDDAS environments. The current methods used to build resilient systems and algorithms are ad-hoc (e.g., randomize memory location or instruction set, change operating system, etc.). In our research, we developed a methodology, rather than ad-hoc methods, to build resilient systems based on three attributes: 1) Redundancy – using physically and/or logical redundant resources; 2) Diversity – using software components that are functionally equivalent but implemented using different programming languages, platforms or algorithms; and 3) Shuffling –randomly changing the lifespan for each version.

We showed that by using the DDDAS paradigm, we can dynamically configure these attributes to make it extremely difficult for attackers to know what resources, programming languages, or operating systems are being used, and thus the attackers cannot succeed in exploiting existing vulnerabilities and cannot apply successful attacks. By dynamically changing these attributes due to detected malicious activities or changes in security policies, we can meet any security and resilience requirements at runtime [1], [2], [3], [4].

2) **Resilient Cloud Services (RCS) and Middleware:** Building diversified software components is typically expensive in terms of development, execution time, and overhead. Our research overcomes this challenge by using cloud services and virtual machines (VMs, each VM runs one diversified software component). In addition, efficient resumption of computations on different platforms after each version lifespan is a challenging research problem. To overcome this problem, we adopted portable checking pointing technique and showed acceptable performance and low overhead on several general applications [3], [4], [5], [6].

3) **Analytical Resilient Modeling:** Quantifying resilience is a challenging research problem, and because of that, it was not well investigated. We developed a resilient modeling approach based on attack surface. In this approach, we used the attack surface to quantify the probability of successful attacks when we use Software Behavior Encryption (SBE) algorithm. The first step is to identify the attack surface by analyzing the software modules and libraries used by the application. For each detected vulnerabilities, we use the The National Institute of Standards and Technology (NIST) Common Vulnerabilities and Exposures (CVE) to estimate the probability of successful attack. By determining the probability of a successful attack that can exploit any of the existing vulnerabilities during the lifespan of one version, we can quantify the resilience of the SBE algorithm against the existing vulnerabilities. We showed that the probability of a successful attack can be reduced to almost zero if we can use three or more diverse versions in the SBE algorithm [7], [8].

2. Introductions

2.1 Intrusion Detection Techniques

Intrusion detection can be broadly classified as signature based and anomaly based systems.

2.1.1 Signature based Intrusion detection systems

A signature based Intrusion Detection System (IDS) uses pattern-matching algorithms to compare network traffic with an extensive library of attack signatures created by human experts [9]. A match indicates a probable attack. These techniques are extremely proficient in detecting known attacks because they can identify an attack as soon as it occurs. However, their foremost limitation is that they cannot detect new attacks or slight variation in known attacks. When a new attack is discovered, it takes time to develop signatures for the attack and deploy them into the existing IDSs. During that period, the attackers exploit the fact that many computers are not protected against their attacks and will continue to be unprotected even after a signature has been found; due to the manual patching of signature, it takes a long period before all computers are patched against the newly discovered attack. Some of the most commonly used signature-based intrusion detection techniques are introduced by SNORT [10], BRO [11], and others [12].

2.1.2 Anomaly-based Intrusion detection

Anomaly-based detection techniques build a model of normal behavior and automatically classify statistically significant deviations from the normal profile as being abnormal [13], [14], [15], [16]. The advantage of this approach is that it is possible to detect unknown attacks. However, there is a potential for having a high rate of false positive alarms generated when the knowledge collected about normal behaviors is inaccurate. Supervised learning approaches for anomaly detection involve training a system on a known set of normal data and testing with a different data set to determine whether the new data is normal. Examples of such techniques are the PI's Anomaly Behavior Analysis (ABA) methodology that has been successfully applied to a wider range of protocols (TCP/IP, DNS, WiFi, HTTP, etc.) [13], [17], [18], [19]. Other anomaly techniques include IDES [20], NIDES [21], EMERALD [22], and SPADE [23]. Other approaches estimate parameters of a probabilistic model over the normal data and compute how well new data fits into the model [24], [25]. Unsupervised techniques include those based on statistical approaches [23], [26], [27], clustering [28], outlier detection schemes [29], [30], [31], [32] and state machines [33] that can detect anomalous behavior without training data. In [34], Shon et al. use a hybrid approach that combines supervised and unsupervised learning mechanisms to perform anomaly detection.

2.2 Security in Cloud Computing

Cloud security suffers from a wide range of attacks such as those that target physical machines as well as the cloud-virtualized environment [35]. Security issues arise from all the different aspects of the cloud such as virtualized environment, service delivery model, customers' data handling, network, and web protocols.

Virtualization is one of the fundamental concepts of the cloud computing. In a cloud system, multiple virtual guest machines share the same resources of a physical host machine. The Virtual Machine Monitor (VMM) is responsible for isolating the VMs from each other while multiplexing the physical resources. There are some known security issues with common VMMs (e.g. Xen, Microsoft HyperV) that can be exploited to threaten cloud services [36]. Hypervisor exploitation [37], [38] is another attack that targets the dependency of cloud computing on the virtualized environment.

Some previous works have presented classifications of the cloud security [39], [40], [41]. In [39], the authors have performed a comprehensive survey on cloud security risks according to known service delivery models. Cloud computing delivers services to the end users through three different delivery models: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). In IaaS, the infrastructure resources such as computation, storage, network, etc. are offered as services. Depending on the type of deployment (public cloud, private cloud, and hybrid cloud), IaaS suffers from varying degrees of security issues. Public clouds pose major risk compared to private clouds. Since cloud systems have a multi-tenant architecture, several resources are shared among users. IaaS services may not be designed to provide strong isolation among tenants, enabling malicious insiders to gain access of legitimate user's data [41]. In PaaS, the service provider offers customers productive platforms that they can use to develop and deploy their own applications on the cloud. Abusive use of APIs could provide a threat to all three service-models [41]. In SaaS, the customers can remotely connect to the cloud to use the provided software applications. Cross-site scripting, access control weaknesses, operating system (OS) and SQL injection flaws, cross-site request forgery, cookie manipulation, hidden field manipulation, insecure storage and insecure configuration are the threats to data stored in a SaaS [42], [43], [44]. Network security issues in SaaS involve network penetration and packet analysis, session management weaknesses, and insecure SSL trust configuration. Also, according to this taxonomy, all security issues pertaining to data locality, data integrity, data segregation, data access, data confidentiality, data breaches, virtualization vulnerabilities, and web application security are applicable to SaaS. Moreover, threats such as Denial of Service (DoS) or Distributed DoS (DDoS) attacks can be a threat to the availability for SaaS. Authentication weaknesses and insecure trust configuration can be security threats to the identity management and sign-on process in SaaS.

Since, in the cloud model, the customers' data reside on third parties' data-centers, data security is a major concern for both cloud consumers and providers and some researchers have addressed data security in their works [45], [46], [47]. Data locality in cloud means that the cloud providers should have the ability to control the data location in order to satisfy the location boundaries of data according to customer's preference. In fact, many organizations or countries have different regulations and limitations about data privacy and data locality. Data integrity is another important cloud security challenge ensuring the trustworthiness of the data during its life cycle. Also, since the data may be replicated in multiple places across cloud's datacenters, any change in the data has to be propagated throughout all replications. Data segregation is another security requirement for cloud since the data from different customers reside at the same location (multi-tenancy). Therefore, the intrusion into a user's data by adjacent users is possible. This kind of intrusions can be performed either by hacking the application or through client code injection (e.g. SQL code injection). Data access is an important cloud security metric. Each customer has his/her own access policy that has to be applied on his/her own data. The cloud access control model should be able to manage access to the data from inside and outside of the customers' organizational boundary. Proper access control mechanism is needed to protect the customers' data from unauthorized users. It also should be able to define accessible part of data for each user. Data confidentiality and privacy are among the major concerns of cloud customers. In fact, by adopting cloud computing, the customers are disclosing their data to the cloud providers. The main concern here is how the cloud providers treat customer's confidential data. Data breaches can also threaten the cloud consumers. Since the customers' data are uploaded to the cloud, any breach in the cloud environment potentially threatens all the customers. This makes the cloud a high value target for outsider attackers. In addition, one of the main security issues in cloud computing is the insider attacks. The insider attacks are considered to be a high-risk threat from current or former employees with the potential access to a huge source of customers' information. With exchange of cloud data between different organizations, the risk of insider attacks increases.

Furthermore, since the cloud services are accessible using Web applications and services, the web protocols security is a major issue in cloud computing. Security holes in the web applications and services create vulnerability to the cloud services [39]. The Open Web Application Security Project has identified Top 10 security risks faced by web applications [48] that can be used to threaten the cloud services (e.g. SQL injection, cross site scripting, etc.).

While various solutions have been proposed to solve cloud security issues [45], [46], [49], [50], there is no comprehensive solution which covers all aspects of cloud security. Most of the offered solutions are partial and apply the detect-response model that fails with time. Some cloud security systems have implemented a recovery-based intrusion tolerant algorithm that enhances the availability and resilience of the cloud services [49]. One security approach focused on hiding the data as a method to increase services' resilience to attacks [49]. Other security focused on efficiently protecting the cloud storage against diverse range of attacks including rollback attacks [47].

Some of the proposed security solutions use risk-based analysis for testing the security of the cloud environment. This risk-based analysis reduces the number of possible misuse cases of the cloud [50].

2.3 Moving Target Defense and Software Diversity

Moving Target Defense (MTD) has been recognized as a game changer approach to build self-defending systems [51]. The MTD technique is based on deliberately introducing spatiotemporal diversity to make the environment resilient to vulnerabilities' exploitations [52], [53]. Diversity can be applied to two entities: the running processes and the execution environment. In spatial diversity [53], multiple replicas of the diversified entity are concurrently invoked in multiple locations. In temporal diversity [52], [53], the entity itself is changing with time, i.e., a more narrow scope of MTD is applied to the entity.

Some works presented a wide range of MTD techniques to continuously change network configurations or parameters, firewall settings, OSs, memory addresses, instruction sets, or application execution environments [54], [55]. For example, in [54], the IP addresses are dynamically changed while maintaining existing connections. One can also randomize the configuration space [56] where the configuration variables of a system are randomized, while ensuring the availability of end to end services. In [57], the authors presented a survey of several software fault tolerance techniques. The fault tolerance techniques that are based on diversity include dual-node redundant operating stations with hardware or software result comparison, recovery block station [58], distributed recovery block with acceptance test [59], voting triple modular redundant computing stations [60], and N-version programming [61]. Also, in [62], several diversity defense techniques in popular OSs were discussed including address space randomization [63], instruction set randomization [54], and data randomization [65].

Some previous works have adopted diversity as a defense technique in a cloud environment. In [66] the authors envision a cloud of clouds architecture, which provides incrementally high levels of security and dependability to cloud infrastructures, in an open, modular, and versatile way. Their architecture employs diversity in deployment of cloud alternatives. However, they do not employ shuffling on these alternatives. In [67], a framework for proactive fault tolerance is discussed that predicts failures in nodes and migrate their processes away from the nodes that are about to fail. In [68], the authors envision a cloud environment with continuous change in system configuration in order to create an unpredictable target for an adversary. To create MTD, they propose to create and operate a large number of replicas, some of which are provided fake inputs to deceive adversaries. They also use diversified replicas for task execution. However, they do not employ shuffling of task versions on each replica. In [69], the authors presented an intrusion tolerant cloud architecture that adopts the method of hybrid fault model, active and passive replicas, state update and transfer, proactive recovery and diversity. This method allows the system for tolerating F faulty replicas in N=2F+1 replicas and ensure that only F+1 active replicas to execute during the intrusion-free stage. The remaining replicas are all put into passive mode, which significantly reduces the resource consuming in cloud platform. However, they do not mention how the state is transferred among diverse replicas.

2.4 Discussion and Comparison

In our approach, we applied spatiotemporal diversity to the processes of the cloud applications and their execution environments. Redundancy in resources is used to run the cloud services and the spatiotemporal diversities are randomly introduced to make it extremely difficult for attackers to figure out the existing vulnerabilities in the currently running applications or their execution environments. The developed approach does not depend on a single programming language; instead it supports using functionally equivalent replicas with diverse implementations to make it harder for attackers to achieve their goals by exploiting design errors or vulnerabilities. Unlike other approaches of MTD for the cloud [53], our approach does not depend on a single compiler or restrict application implementation to a single programming language. The approach also applies the MTD to the environment, i.e., it continuously changes the running VMs and their physical locations [70]. Our experimental results that will be presented later show that our approach can be easily deployed to the cloud and it increases its resilience dramatically.

3. Project Overview

The increased dependence of the U.S. on cyber systems in all Department of Defense (DoD) domains as well as in business, finance, government, and education make them prime targets for cyberattacks due to the fact that profound and catastrophic damage from these attacks might inflict on our economy and all aspects of our life. It is widely recognized that cyber resources and services can be penetrated and exploited. Furthermore, it is widely accepted that the cyber resilient techniques are the most promising solutions to mitigate cyberattacks and change the game to the advantage of the defender over the attacker. The main goal of our project was to use the DDDAS paradigm to develop innovative MTD capabilities that continuously adapt their algorithms based on the current measurements and monitored information. By using DDDAS architecture, we designed a resilient DDDAS (rDDDAS) environment that continuously monitors and analyzes current state of the cyber system and the current configurations of software and hardware resources to discover existing or newly introduced vulnerabilities and anomalies, performs situation awareness and prediction, and applies the appropriate software “behavior encryption” algorithm so that the system tolerates any anomalous event triggered by cyberattacks, malicious faults or accidents. Specifically, the rDDDAS algorithms developed in this project supported the first three Potential Capability Areas (PCAs) identified in the Air Force Technical Horizons report [71]:

PCA1: Inherently Intrusion-Resilient Cyber Systems

- PCA2: Automated Cyber Vulnerability Assessments and Reactions
- PCA3: Decision-Quality Prediction of Behavior

In this project, we developed the rDDDAS capabilities based on MTD concept which is defined as *“Create, evaluate and deploy mechanisms and strategies that are diverse, continually shift, and change over time to increase complexity and costs for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency”* [72]. The developed rDDDAS environment makes it extremely difficult for any attacker to exploit existing vulnerabilities in DDDAS by continuously changing the execution environment. Thus, by the time an attacker studies a DDDAS’ vulnerability to construct an attack and then launch it, the DDDAS execution environment has already changed to a new environment, thereby rendering the attack ineffective. The rDDDAS environment utilized the following capabilities:

Replication: It is commonly used in fault tolerance techniques [60] in order to continue to operate successfully in spite of software or hardware faults. In our approach, we combined the N-version programming [61] with hardware and VM redundancy such that each cloud application task runs on different physical nodes as well as on different VMs in the cloud infrastructure.

Diversity and Automatic Checkpointing: This capability enabled us to generate multiple functionally-equivalent, behaviorally-different software versions (e.g., each software task can have multiple versions, where each version can be a different algorithm implemented in different programming language (e.g., C, Java, C++, etc.) that can run on different computing systems. We used the Compiler for Portable Checkpointing (CPPC) [73] to capture the current state of the cloud application such that it can be resumed on different cloud environments.

Software Behavior Encryption (SBE): SBE uses spatiotemporal behavior obfuscation to make active software components change their implementation versions and resources continuously and, consequently, evade attackers. This approach significantly reduced the ability of an attacker to disrupt the normal operations of a cloud application. Also, it allows for adjusting the resilience level by dynamically increasing or decreasing the shuffling rate and tasks’ versions and their execution environments. A major advantage of this approach is that the dynamic change in the execution environment can hide the software flaws that would otherwise be exploited by a cyberattacker.

Autonomic Management (AM): The primary task of the AM is to support dynamic decision making among the various components such that the cloud resources and services are dynamically configured to effectively exploit the current state of the cloud system and meet the application security requirements that might change at runtime.

Our solution approach to develop rDDDAS is shown in Figure 1 and is based on the following capabilities: Online Monitoring and Measurement, Modeling, Analysis, and Prediction, SBE, and Autonomic Management. In what follows, we present our research results in developing the rDDDAS environment.

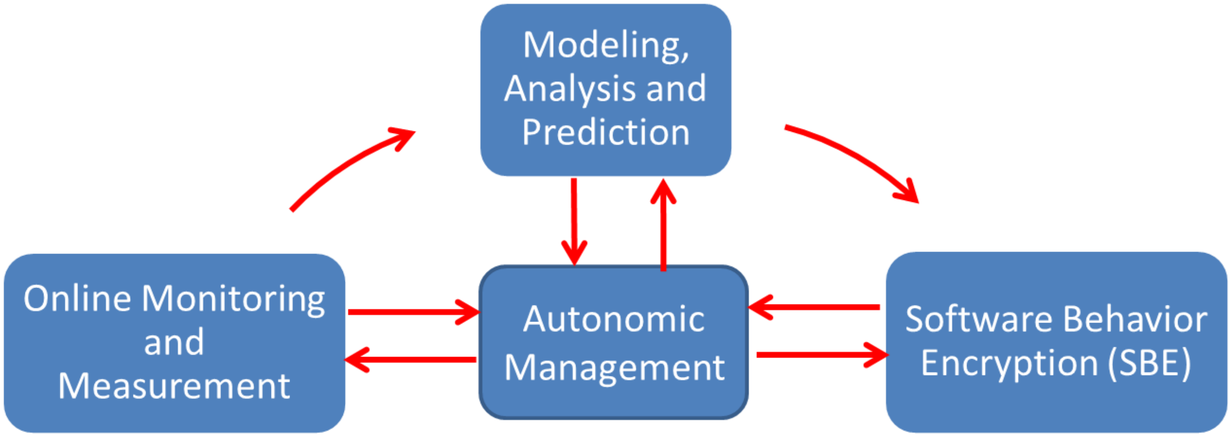

Modeling, Analysis and Prediction
Online Monitoring and Measurement
Autonomic Management
Software Behavior Encryption (SBE)

Figure 1: Resilient DDDAS (rDDDAS) architecture

4. Project Research Results

4.1 Software Behavior Encryption (SBE)

The SBE algorithm hides (analogous to data encryption) the execution environment by dynamically changing the execution sequence of task variants after each execution phase. The dynamic change in software behavior makes it extremely difficult for an attacker to identify the possible flaws of the executing variant (task versions). The decisions regarding when to shuffle the current variant, the shuffling frequency, and the variant selection for the next shuffle are guided by a continuous monitoring and analysis of current execution state of cloud applications and the desired resilience requirements.

As shown in Figure 2, any attack will go through at least three phases: probing, constructing, and launching phases. If the environment stays static as it is typically now, the attacker has plenty of time to identify existing vulnerabilities that can be exploited and consequently can succeed in launching the attack (Successful attack scenario in Figure 2). However, if the life cycle for any version is much shorter than the time it takes for the attacker to launch the attack it will not be able to succeed in exploiting existing vulnerabilities (Thwarted Attack scenario)

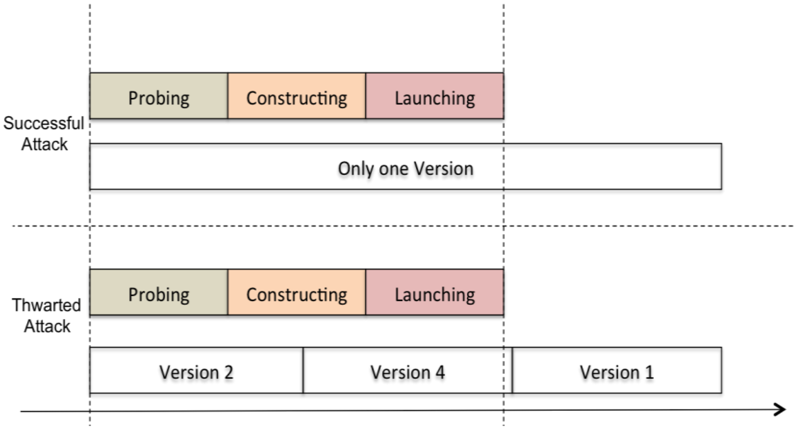


Figure 2. Attack Window for in SBE algorithm

Figure 3 shows an example on how SBE algorithm can be implemented to hide the execution of one task (Task A) that can run sequentially as three subtasks TA_1, TA_2 and TA_3 that run in three consecutive phases, respectively. During phase 1, we execute version 3 of subtask TA_1, version 1 of subtask TA_2 during Phase 2, and version 1 subtask TA_3 during Phase 3 (see Figure 3).

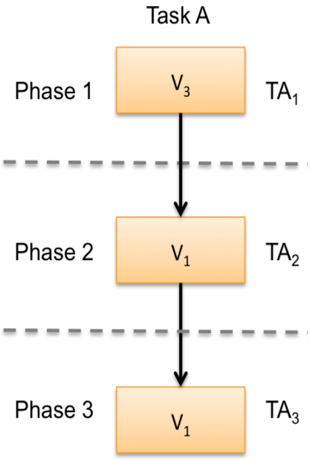


Figure 3: SBE example

In addition to the shuffling of the execution of the task variants, we also apply hardware redundancy and software diversity to the implementation of the application tasks. The concept of design diversity is commonly used in software fault tolerance techniques in order to continue to operate successfully in spite of software design faults. In our SBE implementation approach, we combine N-version programming [61] with hardware/software redundancy techniques. The multi-version implementation will prevent adversarial attacks from exploiting the monoculture problem that allows attackers to succeed in infecting million instances of software systems and/or applications that have the same vulnerability [52].

Based on the resilience requirements, we can use multiple spatiotemporal diversity techniques (e.g., at application/ level, task level, OS level, and resource level). In Figure 4, we show how to apply the SBE algorithm to an application that is implemented in three phases. For each task, we created 6 functionally equivalent versions on three physical nodes. For example, during Phase 1, version 2 of the application runs in a Linux environment on Node 1, while Version 3 runs in a Windows environment on Node 2, and Version 6 runs in Mac environment on Node 3. All nodes receive the same input and act on it in parallel so any attack-free version can be used to pass the results to the second phase. The anomaly behavior analysis approach developed by our research team has been used to ensure that the operations of each task are completed correctly at the end of each execution phase [17], [18], [74], [77]; by using normal runtime models we can detect any malicious changes in the execution environment, task variables, memory access range etc.

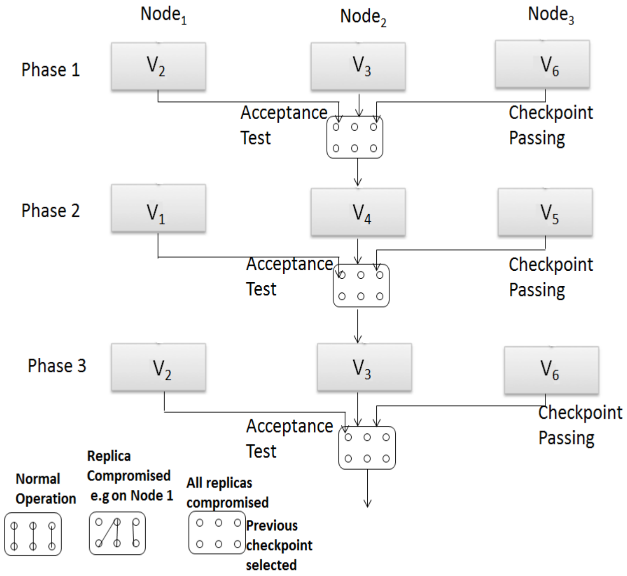


Figure 4: SBE application execution with three phases.

4.1.1 Portable Check Pointing

To support the capability to resume the execution of the functionally equivalent variants on different platforms in another phase or when they recover from attacks or faults, we adopted Compiler for Portable Checkpointing (CPPC) [73], [78] technique and used it in the SBE algorithm. Checkpointing is widely used method to recover from fault once it is detected as in fault-tolerance computing [56], [57]. It periodically saves the computation state to a stable storage so that the application execution can be resumed by restoring such state.

CPPC is a checkpointing tool focused on the insertion of fault tolerance into long-running applications. CPPC allows for execution to restart on different architectures and/or OSs. It also attempts to optimize the amount of data saved to disk for improved efficiency and data transfers over the network. CPPC provides portable restart of applications in heterogeneous environments. Generated state files can be used to restart the computation on an architecture or an OS different from the one that generated the file. To achieve this portability, CPPC-generated state files do not contain architecture-dependent state. Rather, this state is recovered during a restart by re-executing the code that created the opaque state in the original run. In order to achieve portability of actual user data, CPPC uses a checkpoint file format based on HDF5, a data format and associated API for the portable transfer of scientific data between computers [79]. Portable offsets are used to store pointers in a way that preserves aliasing relationships throughout application restarts. Together, these techniques enable restart on different architectures.

Figure 5 illustrates the global process for checkpoint generation in CPPC. The first step is to select suitable checkpoint locations and insert the appropriate function call. Afterwards, the application data that needs to be stored and recovered at each checkpoint to achieve consistent application restart has to be selected. Finally, the application code needs to be modified; inserting control flow constructs to correctly restart the application after a failure. In order to free the user from these tasks, CPPC includes a source-to-source compiler that performs the required code analysis and transformations and outputs a fault-tolerant version of the input code.

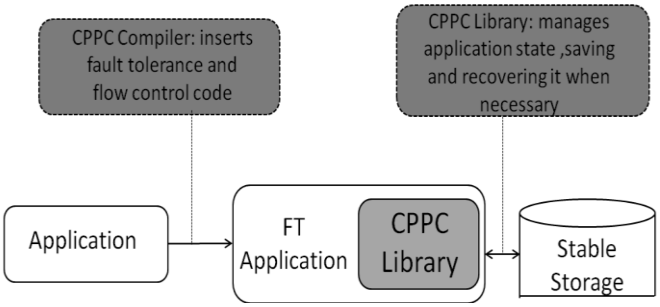

CPPC Compiler: inserts fault tolerance and flow control code
CPPC Library: manages application state ,saving and recovering it when necessary
Application
FT Application
CPPC Library
Stable Storage

Figure 5: Compiler for Portable Checkpoint generation

Checkpoint locations are automatically selected by identifying sections of code that take a long time to execute, where checkpoints are needed to guarantee execution progress in the presence of failures. The time to compute a given section of code cannot be accurately predicted at compile time without the knowledge of the computing platform, input data, and binary code to be executed. For this reason, heuristic analyses are used. The compiler discards any code location that is not inside a loop, and ranks all loop nests in the code using computational metrics such as the number of memory accesses and statements executed inside the given loop. A call to the checkpoint routine is then inserted in the selected loop nests. Under certain conditions, the CPPC compiler is able to statically analyze the communication patterns of a parallel application, enabling CPPC to checkpoint MPI codes. A full description of the checkpoint insertion heuristics and how parallel processes are coordinated can be found in [80].

Once checkpoint calls are inserted, and in order to identify the data that are required when restarting the application from each checkpoint location, the CPPC compiler performs an inter-procedural liveness analysis. Live variables at each checkpoint location are selected for storage into state files. CPPC is an open-source tool and is available at http://cppc.des.udc.es under GPL license.

4.1.2 SBE Algorithm

We have successfully designed and implemented a general autonomic computing environment that has been used to implement the AM module [82], [83]. By adopting the autonomic architecture shown in Figure 6, we implement the AM services using two software modules: Observer and Controller modules. The Observer module monitors and analyzes the current state of the managed cyber resources or services. The Controller module is delegated to manage the cloud operations and enforce the resilient operational policies. In fact, the Observer and Controller pair provides a unified management interface to support the AM's self-management services by continuously monitoring and analyzing current cloud system conditions in order to select dynamically the appropriate plan to correct or remove anomalous conditions once they are detected and/or predicted.

Figure 7 describes the algorithm for managing SBE's task replicas. In step 1, the AM initializes the SBE to generate the list of task's replicas with the required phase versions. The AM controller uses the list generated from the SBE to set the policies that manage and control the phases of all the replicas (step 2). The observer then updates its sensors based on the SBE output, and it starts monitoring the execution of different replicas when they run (steps 3 & 4). The AM starts running all the replicas (steps 7, 8, and 9). The observer monitors and checks the state of the replicas continuously (step 12). If any of the replicas finished the phase successfully (i.e., error or attack free), then all the other replicas are stopped for this phase and the successful output of this phase is used as an input for all the replicas in the next phase (steps 13 to 17). While monitoring and checking the replicas, if any of the replicas is behaving abnormally, then that replica is stopped and the SBE algorithm is invoked again to generate new phases/versions list for that task replica. The new output of the SBE is then used to launch the replica from the current phase using the successful output of the previous phase (steps 18 to 22). Finally, the output of the first successful replica finishing the last phase is used as the output of the task.

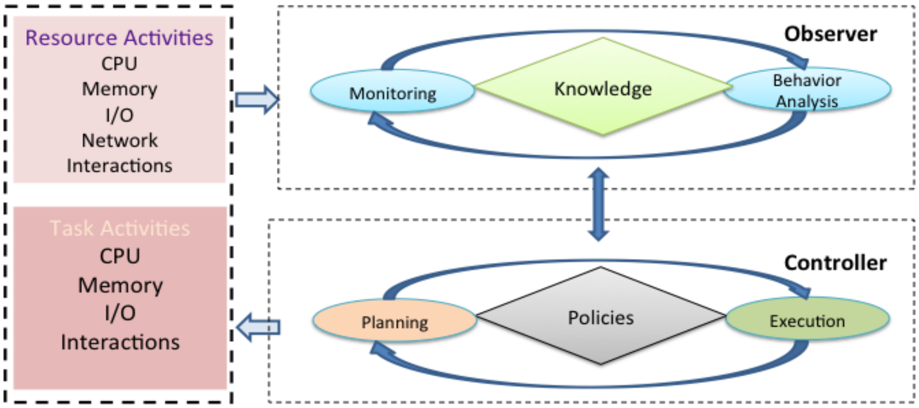
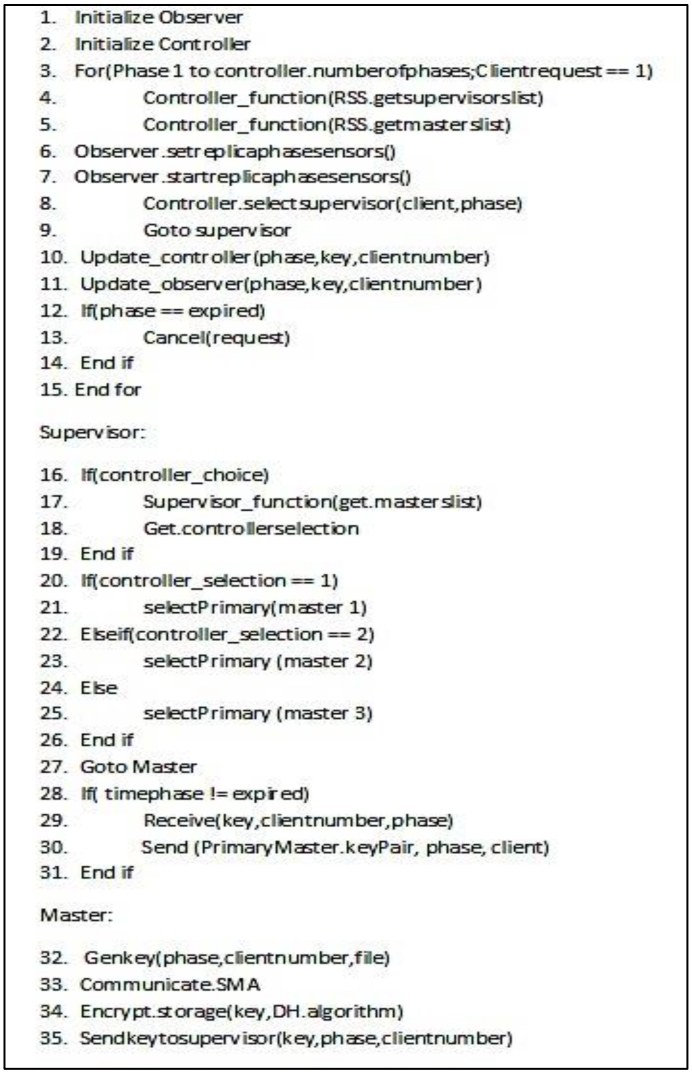


Figure 6: Autonomic management module Architecture



```
Initialize Observer
Initialize Controller
For(Phase 1 to controller.numberofphases;Clientrequest == 1)
        Controller_function(RSS.getsupervisorslist)
        Controller_function(RSS.getmasterslist)
Observer.setreplicaphasesensors()
Observer.startreplicaphasesensors()
        Controller.selectsupervisor(client,phase)
        Goto supervisor
Update_controller(phase,key,clientnumber)
Update_observer(phase,key,clientnumber)
If(phase == expired)
        Cancel(request)
End if
End for

Supervisor:

If(controller_choice)
        Supervisor_function(get.masterslist)
        Get.controllerselection
End if
If(controller_selection == 1)
        selectPrimary(master 1)
Elseif(controller_selection == 2)
        selectPrimary (master 2)
Else
        selectPrimary (master 3)
End if
Goto Master
If( timephase != expired)
        Receive(key,clientnumber,phase)
        Send (PrimaryMaster.keyPair, phase, client)
End if

Master:

Genkey(phase,clientnumber,file)
Communicate.SMA
Encrypt.storage(key,DH.algorithm)
Sendkeytosupervisor(key,phase,clientnumber)
```

Figure 7: SBE Algorithm.

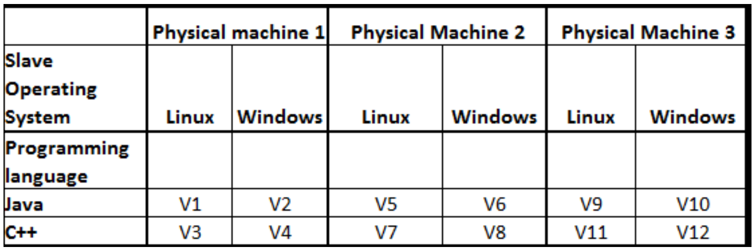
4.2 Resilient Applications and Performance Evaluation

In this section, we show that the SBE algorithm can be used to develop resilient applications or routines that can be used as modules in the development of rDDDAS environments.

4.2.1 Resilient MapReduce Application

MapReduce is widely used as a powerful parallel data processing model to solve a wide range of large-scale computing problems [85]. With the MapReduce programming model, programmers need to specify two functions: Map and Reduce. The Map function receives a key/value pair as input and generates intermediate key/value pairs to be further processed. The Reduce function merges all the intermediate key/value pairs associated with the same (intermediate) key and then generates the final output. There are three main roles: the master, mappers, and reducers. The single master acts as the coordinator responsible for task scheduling, job management, etc. MapReduce is built upon a distributed file system (DFS) which provides distributed storage. The input data is split into a set of map (M) blocks, which will be read by M mappers through DFS I/O. Each mapper will process the data by parsing the key/value pair and then generate the intermediate result that is stored in its local file system. The intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together. The locations of the intermediate results will be sent to the master who notifies the reducers to prepare to receive the intermediate results as their input. Reducers then use Remote Procedure Call (RPC) to read data from mappers. The user defined reduce function is then applied to the sorted data; basically, key pairs with the same key will be reduced depending on the user defined reduce function. Finally, the output will be written to DFS.

Apache Hadoop is an open source implementation of the MapReduce framework [86] and is used in our experimental results to evaluate our system for the MapReduce application. We have chosen Oracle Virtualbox [87] as the virtualization software. To maintain consistency with the MapReduce parlance defined in [85] we will refer to each physical host machine as master and each guest machine as slave (refer to Figure 8). To prevent any single point of failure, each guest machine is configured to run in a single node cluster. The MapReduce Wordcount [88] program is available on each slave in C++ and Java. Thus, the combination of <physical machine, OS, and programming language> represents a single version. Figure 8 provides details about the application diverse versions used in our implementation.



	Physical machine 1		Physical Machine 2		Physical Machine 3	
Slave Operating System	Linux	Windows	Linux	Windows	Linux	Windows
Programming language						
Java	V1	V2	V5	V6	V9	V10
C++	V3	V4	V7	V8	V11	V12

Figure 7: Versions used for the MapReduce Application

The Map/Reduce application in our experiment is divided into three phases as follows:

- Phase 1: First Map function
- Phase 2: Second Map function
- Phase 3: Final Map/Reduce function.

The output of the previous phase is used as input in the next phase (e.g. the Phase 1 output is input for Phase 2 and the output of Phase 2 is input for Phase 3). During runtime, the application execution is performed in parallel on each of the three machines. Also, at the beginning of each phase, each master runs a local shuffler program to determine the version to run at the current phase. For this experiment, we have used a random number generator to determine the version that will run on each machine. At the end of each phase, the three masters run local acceptance tests. If any master’s acceptance test fails, its output is discarded and the output is retrieved from another master.

Figure 9 shows an example of SBE in our experiment. At the beginning of Phase 1, masters 1, 2, and 3 run a random number generator and select versions V1, V8, and V10 respectively. After completion of the first Map on each physical machine, the output is checked for correctness by the acceptance test criteria. If this test fails, the master selects the output of phase 1 from other masters and the first result that passes the acceptance test will be selected for the next phase of the application execution. Similar actions are performed by each master in phases 2 and 3.

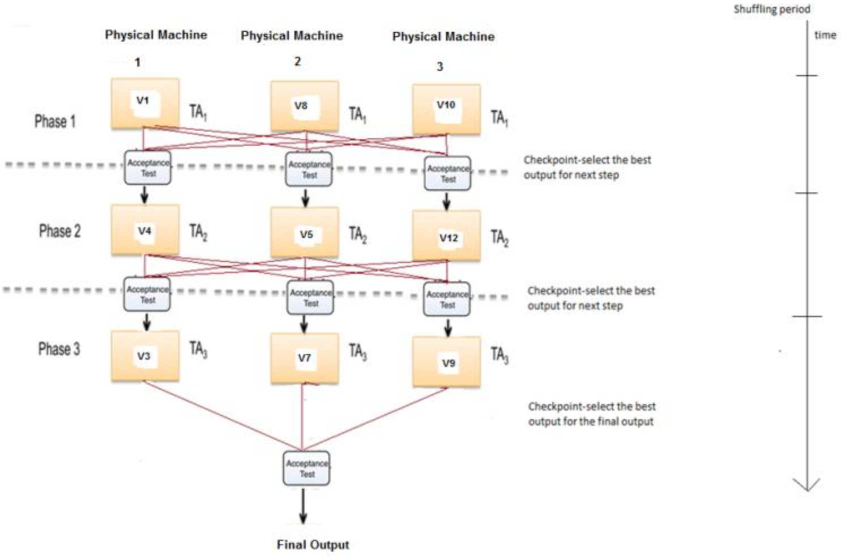
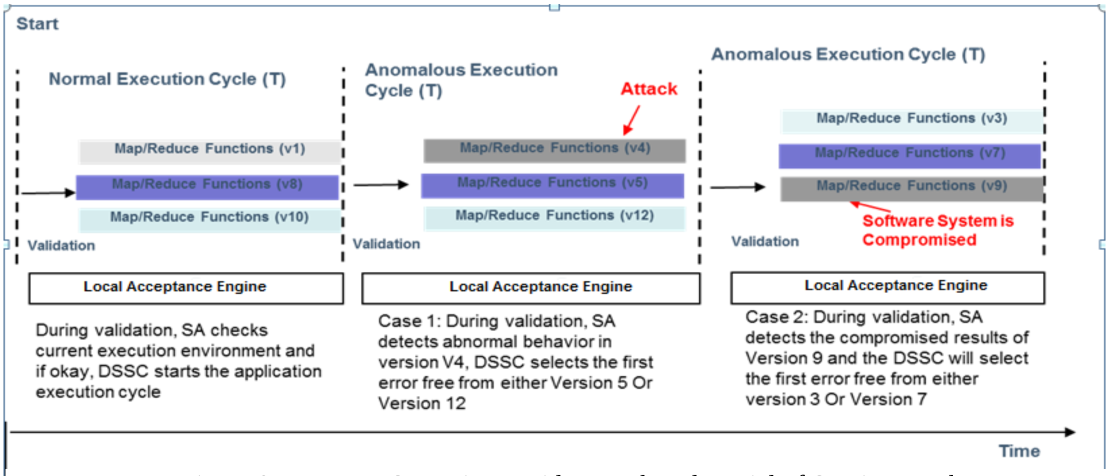


Figure 8: Example of SBE used in the MapReduce experiment

We have evaluated our approach for the two cases shown in Figure 10:



Start
Normal Execution Cycle (T)
Anomalous Execution Cycle (T)
Attack
Anomalous Execution Cycle (T)
Map/Reduce Functions (v1)
Map/Reduce Functions (v8)
Map/Reduce Functions (v10)
Map/Reduce Functions (v4)
Map/Reduce Functions (v5)
Map/Reduce Functions (v12)
Map/Reduce Functions (v3)
Map/Reduce Functions (v7)
Map/Reduce Functions (v9)
Software System is Compromised
Validation
Validation
Validation
Local Acceptance Engine
Local Acceptance Engine
Local Acceptance Engine
Time

During validation, SA checks current execution environment and if okay, DSSC starts the application execution cycle

Case 1: During validation, SA detects abnormal behavior in version V4, DSSC selects the first error free from either Version 5 Or Version 12

Case 2: During validation, SA detects the compromised results of Version 9 and the DSSC will select the first error free from either version 3 Or Version 7

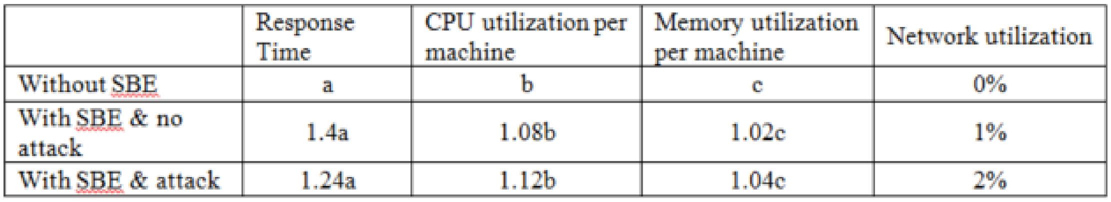
Figure 9: Test case Scenarios- Insider attack and Denial of Service attack

4.2.1.1 Case 1: Resilience against Denial of Service Attacks

In this scenario, we launched a DoS attack on one of the machines used to run the Map/Reduce application. The SBE service was able to successfully detect and tolerate the DoS attack. Although the DoS attack affected the attacked physical machine and increased its response time by 23%, the response time of the application with and without attack remained the same as we took the output from the other physical machine in such a case. In our experiments, we experimented an overhead of 14% in response time by our approach which is due to the additional management compared to the single non-resilient approach.

4.2.1.2 Case 2: Resilience against Insider Attacks

In this case, one of the machines (the fastest physical machine) is compromised by an insider attack and the computations running on that machine were changed by the internal attacker. Similar to the previous case, the application continued to operate normally in spite of the insider attack because the results from the compromised machine were ignored and the results from other versions (4 and 12) were used instead. The performance impacts and overhead on the application performance are shown in Figure 11.



	Response Time	CPU utilization per machine	Memory utilization per machine	Network utilization
Without SBE	a	b	c	0%
With SBE & no attack	1.4a	1.08b	1.02c	1%
With SBE & attack	1.24a	1.12b	1.04c	2%

Figure 10: MapReduce Result Summary

As shown in Figure 11, the average response time using the Resilient Cloud Application Services approach increases by 14% (without attack) and 24% (with attack).

4.2.2 Resilient Jacobi’s Iterative Linear Equation Solver

Linear equations are used to solve a wide range of real world scientific and engineering problems. The Jacobi technique is an iterative technique for solving a set of linear equations under two assumptions [89]:

- The system given by Ax=B has a unique solution
- The co-efficient matrix A has no zeroes on its diagonal.

To solve a set of n equations, we solve the first equation for x1, second equation for x2 as follows: We first make an initial assumption of the values of x. We then substitute these values into the right hand side of the set of equations. This completes the first iteration. This process is repeated until convergence is reached on the values of x.

To evaluate the SBE algorithm, we used a testbed based on the IBM BladeCenter HS22 Private Cloud [90] at University of Arizona’s Centre for Cloud and Autonomic Computing. The implementation runs on a three node cluster each hosting two VMs. One of these VMs has a Windows based OS, while the other one has a Linux based OS. We have used VMware vSphere5 [91] as the virtualization software (See Figure 12).

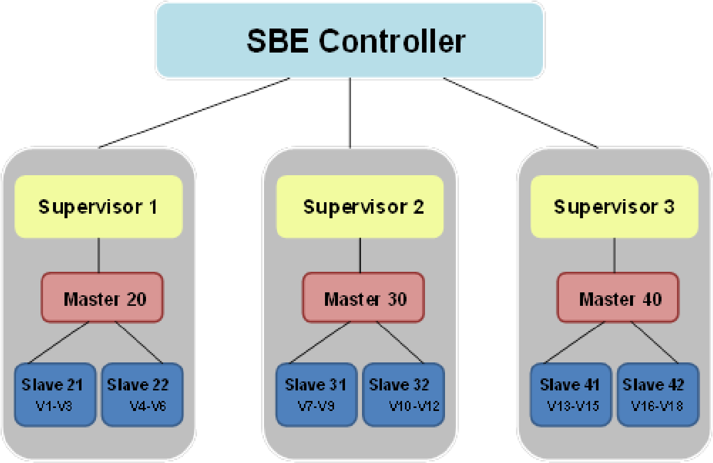
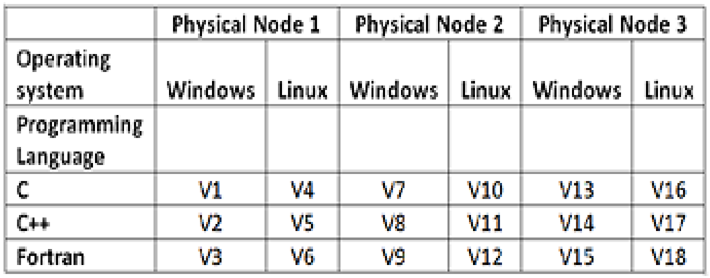


Figure 11: Jacobi's Linear Equation Solver: Testbed Setup

The Jacobi Algorithm described above has been implemented in C, C++, and Fortran, thus creating different versions (see Figure 13).



	Physical Node 1		Physical Node 2		Physical Node 3	
Operating system	Windows	Linux	Windows	Linux	Windows	Linux
Programming Language						
C	V1	V4	V7	V10	V13	V16
C++	V2	V5	V8	V11	V14	V17
Fortran	V3	V6	V9	V12	V15	V18

Figure 12: Versions used in Application 2

Figure 14 provides an example of the SBE used in our experiment.

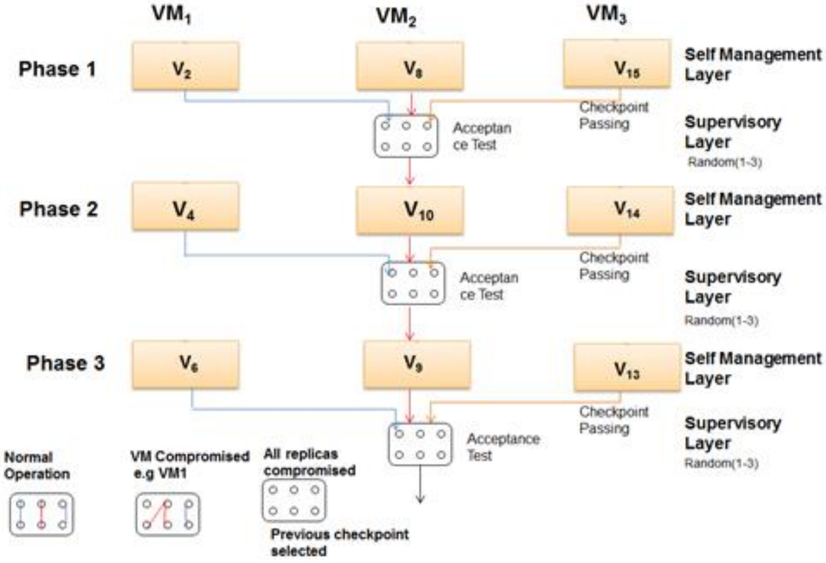


Figure 13: Example of SBE used in Application 2

In the beginning, the SBE controller randomly selects a supervisor from a set of three supervisors (one supervisor on each physical node). This supervisor then randomly selects a phase timer for that phase. The master machines on each physical node randomly select the versions to be run on each node. Checkpoints are continuously stored on a master machine on each physical node. At the expiration of the phase timer, the last checkpoint from each of the three masters is passed onto the supervisor machine (Steps 12-15 in the algorithm shown in Figure 7). An acceptance test is run on each of these checkpoints. This test checks for the following properties: a) The solution is within a range, b) The memory utilization of the program is within a normal range, and c) The variable values after subsequent iterations are not too divergent. The latest checkpoint that passes the acceptance test is selected as the output of this phase. For example, if the checkpoints received from the masters have completed iteration 5, 7, and 8, respectively and if they all pass the acceptance test, the checkpoint which has completed the 8th iteration is selected as the output for this stage and the input for the next phase (Steps 16-21 in Figure 7). At the beginning of the next stage, a new supervisor is selected randomly and the above process is repeated until the final output is received.

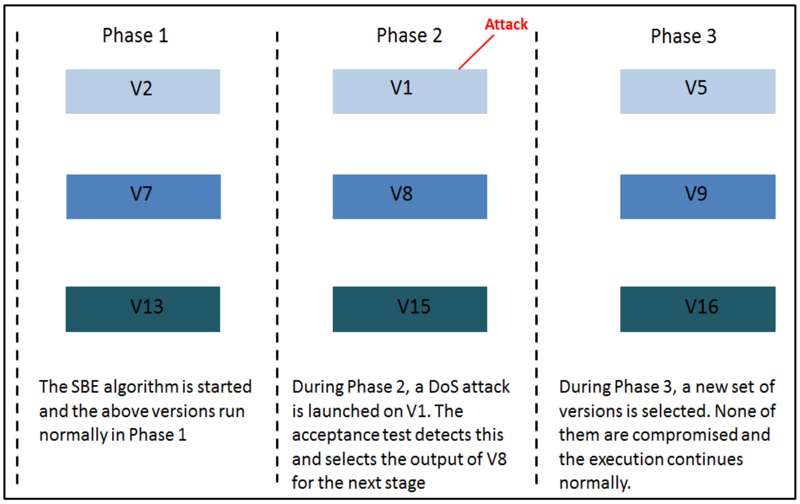
Table I summarizes the overhead in terms of the execution time and overhead percentage for five programs with a normal execution time ranging from 200 seconds to 3600 seconds respectively. The overhead is given as a function of the number of phases selected to run the application.

Table 1: Overhead in Application 2

	Execution time with SBE in seconds					
Execution Time in seconds without SBE	2 phases		3 phases		4 phases	
	Time	OH	Time	OH	Time	OH
200	218	9%	248	24%	276	38%
800	838	5%	890	11%	988	24%
1500	1568	5%	1624	8%	1663	11%
3600	3671	2%	3847	7%	3890	8%

We calculated the overhead as the additional time taken with our algorithm compared to running the application without SBE. As shown in Table 1, for programs with higher execution times, the overhead due to SBE reduces significantly. For example, for a program with execution time of 3600 seconds, the overhead percentage for 3 phases is 7%. The number of phases to run each application can be chosen such that it meets the performance and resilient requirements of the application.

In evaluating the resilience of this application, the following attack scenarios are launched against the application execution. Figure 15 illustrates Scenario 1:



The SBE algorithm is started and the above versions run normally in Phase 1

During Phase 2, a DoS attack is launched on V1. The acceptance test detects this and selects the output of V8 for the next stage

During Phase 3, a new set of versions is selected. None of them are compromised and the execution continues normally.

Figure 14: Attack Scenario 1

1. DoS Attack (Attack Scenario 1).

We launched a DoS attack on the Windows machine running version V1 during Phase 2 using the *mprime* library [92] for memory DoS attack. As a result of the DoS attack, V1 execution was very slow. The acceptance test detected that the checkpoints received from the other two versions were faster and accurate. Hence the checkpoint from another machine was selected for the output of this phase and the application continued to operate normally in spite of the DoS attack.

2. Insider Attack (Attack Scenario 2).

As seen in Figure 12 there are three supervisors that directly communicate with the SBE controller. Only one randomly selected supervisor is active in any given phase. In Scenario 2, in the beginning of the execution, we compromised Supervisor 2 by destroying all the Supervisor services running on it. During the phase when Supervisor 2 was selected, the acceptance test unit on the controller detected that the Supervisor code is not running and selected another Supervisor.

3. Compromising Two VMs (Attack Scenario 3):

In this attack scenario, we compromised the Linux VM on physical machine 2 and Windows VM on physical machine 3 by replacing all code versions on these machines with other programs. The acceptance test carried out by the supervisor detected that the output is irregular and the output from the version running correctly on physical machine 1 was selected. Hence, the application continued to operate normally in spite of hijacking two physical machines.

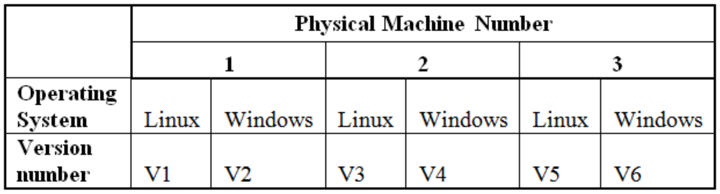
4.2.3 MiBench Benchmarks

The MiBench Benchmarks [93] consist of C programs from six categories each targeting a specific area of the embedded market. We used the following applications from the MiBench benchmark suite:

Basicmath (Automotive and Industrial category): This program performs mathematical calculations like cubic function solving, integer square root, and angle conversions from degrees to radians are all necessary calculations for calculating road speed or other vector values.

Dijkstra's algorithm (Network category): This program constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of Dijkstra's algorithm.

For each of the above available C programs, we used diversity in operating systems to have a total of 6 versions. The versions used are shown in Figure 16. We calculated the overhead of our Resilient Application Services architecture for different number of iterations of the above mentioned benchmarks. The results are presented in Figures 17 and 18. As the experimental results present (Figures 17 and 18), the overhead of our algorithm decreases as program size increases.



	Physical Machine Number					
	1		2		3	
Operating System	Linux	Windows	Linux	Windows	Linux	Windows
Version number	V1	V2	V3	V4	V5	V6

Figure 15: Versions used with the MiBench suite

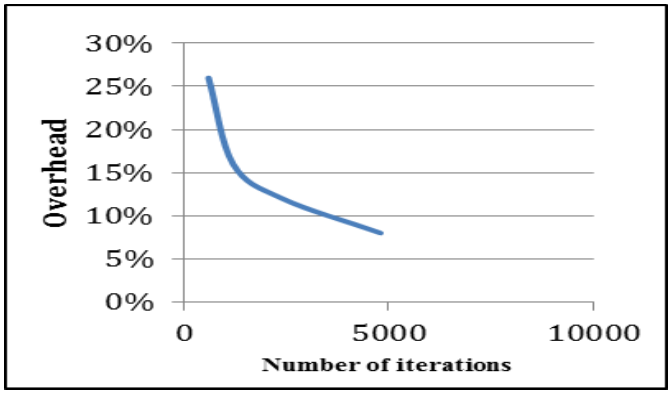


Figure 16: Basicmath - Overhead for SBE with three phases

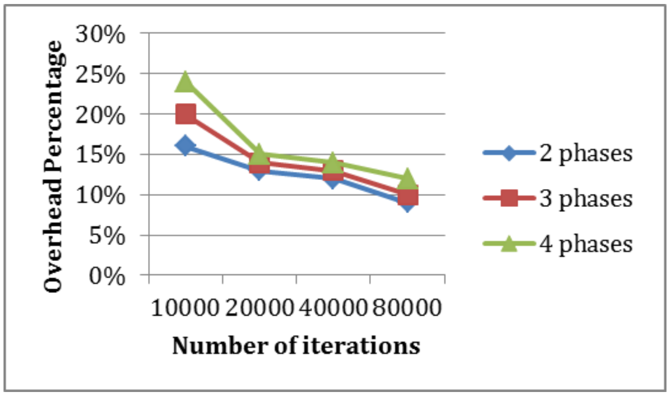
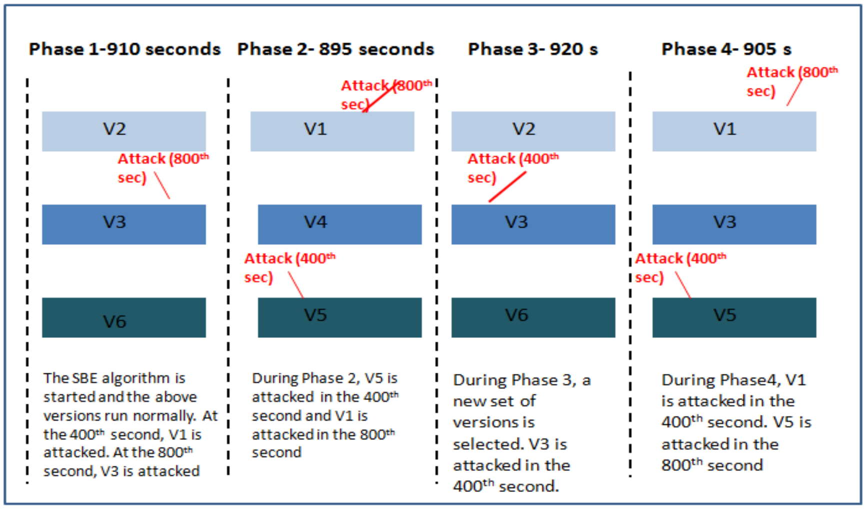


Figure 17: Dijkstra's Algorithm - Overhead for SBE with three phases

We also evaluated the following attack scenario on the Dijkstra program: We first ran the Dijkstra program without SBE and without any attack. We named the output file as 'A'. We then set the SBE phase timer with an average of 900 seconds, i.e. the interval between the version changes had an average of 900 seconds. We also configured an attack machine which launched insider attacks on V1 after the first 400 seconds, V3 after the second interval of 400 seconds, V5 after the third interval of 400 seconds, and so on. We did this for an SBE algorithm with 4 phases. The output received after this was named 'B'. On comparison, A and B were the same. Thus, the attacks were tolerated by our Resilient Cloud Application Services. The attack scenario is illustrated in Figure 19.



The SBE algorithm is started and the above versions run normally. At the 400th second, V1 is attacked. At the 800th second, V3 is attacked

During Phase 2, V5 is attacked in the 400th second and V1 is attacked in the 800th second

During Phase 3, a new set of versions is selected. V3 is attacked in the 400th second.

During Phase4, V1 is attacked in the 400th second. V5 is attacked in the 800th second

Figure 18: Attack Scenario-Application 3

4.3 Resilience Cloud Storage Services

With the advancements in networking and cloud computing technologies, there is an increasing demand for the computational and storage resources. The world's information is doubling every two years [94], more than 300 hours of video content is uploaded to YouTube every minute [95]. Hence, the need for the cloud storage is increasing dramatically. A recent report released by IHS says that an uninterrupted double-digit growth in cloud storage subscription anticipated to follow until at least 2017 [96]. The average annual run rate of IP traffic by the end of 2015 is estimated as 966 Exabyte worldwide [97]. 30% of Vendors use cloud services for storage [98].

Cloud storage has become a business solution for remote storage and data backup as it offers infinite storage space for clients (enterprises/individuals) in a pay-as-you-go manner [99], [100]. For example, Ericsson, a major provider of technology and services to telecom operators, uses Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), and the Right scale Cloud Management Platform for provisioning and auto-scale functionality. Similarly, Shaw Media uses Amazon Web Services (AWS) to improve uptime for its high-traffic websites and also to implement a disaster recovery strategy that resulted in a $1.8 million saving; the cost for a establishing a second physical site. There is an increased interest and deployment of cloud services that use Amazon web services for many services (application, backup and storage, computation, networking) [101]. On the other hand, cloud storage tools like Dropbox, cloudme.com, etc. allow individuals to store their data on the cloud and access it from any computer or mobile device with Internet access. Devices with limited storage like mobiles and tablets prompt users to store their audio/video files on cloud.

According to the Future of Cloud Computing Survey 2011, the main inhibitor to cloud adoption is security [102]. 43% of companies globally currently using a cloud computing service reported a data security lapse or issue with the cloud service their company is using within the last 12 months [103]. 15% of the data centers do not have data backup and recovery plans [104]. The cost of a datacenter outage is calculated as Average of $505,502 per incident [105]. The biggest problem in the adoption of cloud storage is the concern over the confidentiality and integrity of their data [106].

In order to solve the data storage resilience in cloud systems, we developed and evaluated a Resilient Cloud Storage Service (RCSS) that will overcome the security and privacy issues. The RCSS architecture solves two main security problems: 1) Access control that ensures that only authorized users can access data in cloud systems; and 2) Secure communications that prevent any data leakage while the data is in transit. Our experimental results and evaluation of our approach shows that also 50% improvement in performance can be achieved along with secured data services by using a reduced key length of 512 bits.

4.3.1 Resilient Cloud Storage Service Architecture

The resilient cloud storage services are implemented as shown in Figure 20. When a client requests to use the cloud data storage services such as for reading, writing/uploading a file, the Self-Management module (SMM) initiates the secure communication by checking the authentication of the client. The CA certificates [111] are used to verify both the client and the SMM. At this point if the authentication fails, the client is added to the blocked list by the SMM until the client authenticity is verified. The secure communication between client and the cloud storage is implemented in three steps. CA certificate verification is the first step while the Diffie-Hellman (DH) key exchange protocol and key hopping with file partitioning are the second and third steps, respectively. The SMM initiates the DH key [108] generation algorithm between the Storage Management Agent (SMA) and the SMM. Using the DH key exchange protocol, the public and private key pair is generated. Additionally, CA certificate verification is applied. At this point, the client will be sure about the sender of the key and also the man-in-the-middle attack will be extremely difficult. Once the key is received by the client, the communication between the client and the data server is encrypted in two layers.

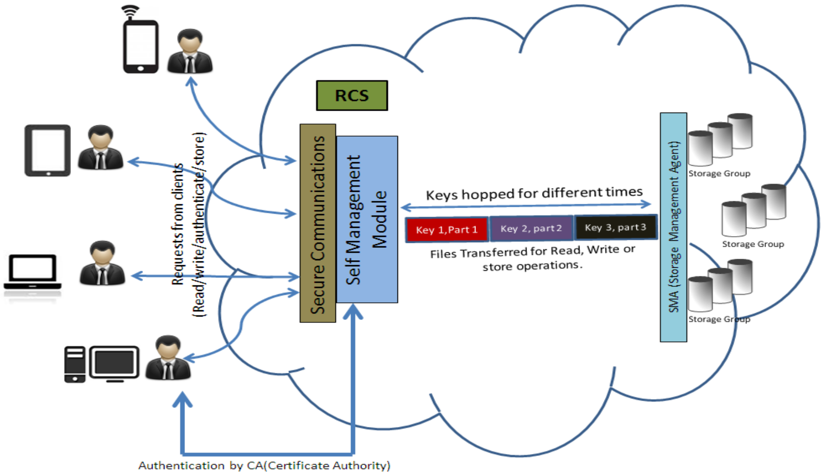

RCS
Secure Communications
Self Management Module
Requests from clients (Read/write/authenticate/store)
Keys hopped for different times
Key 1,Part 1
Key 2, part2
Key 3, part3
Files Transferred for Read, Write or store operations.
SMA (Storage Management Agent)
Storage Group
Storage Group
Storage Group
Authentication by CA(Certificate Authority)

Figure 19: Architecture diagram of RCSS

In the first layer, the files, which are transmitted between the client and the server, are divided into parts and each part is encrypted using DES (Data Encryption Standard) algorithm [109]. These DES keys used in encryption should be known by the data server for decryption, i.e. these keys should be sent to the server. In order to make sure that the DES key is not compromised during the transmission between the client and the data server, the DES key is sent encrypted using RSA algorithm, using the public and private key pair generated with the DH key exchange protocol.

For an attacker to successfully attack the system and steal the data in transit at a particular time window, the attacker needs to know the exact file part and the DES key that is used to encrypt that specific file part and the RSA key that is used to encrypt the DES key. In the worst case, even if the attacker finds out all the required variables (which is nearly impossible), the attacker will be able to see only the data in that particular time window since in the next time window, all keys will change again.

In our approach, we prefer using a sequence of random shorter keys where each key will be active for a random period of time (determined by the SMM) in a similar manner to frequency hopping in wireless networks [110]. When a small key is used, the time window should be small and with multiple hopping in order to make the system secure and resilient to attacks; the attackers will have less time to figure out the key and by the time they might be able to discover it, the key will be changed. This approach will reduce the overhead introduced from the encryption using large keys.

4.3.2 Secure Communications

When a client wants to access the cloud services, the SMM starts a timer and initiates the DH key generation protocol between the client and the SMM. In order to avoid the man-in-the middle attack, the server certificates are verified by the client to make sure that it is receiving the correct key from the correct sender [111]. This key is generated by the client to prove its identity to the cloud provider. The access control list is then updated and communication starts between the client and cloud system. The SMM manages the channel encryption and the key hopping for a randomly selected time windows. The channel is encrypted using DES (Data Encryption Standard) in CFB64 (Cipher Feedback) mode [112]. In this CFB mode, the first 8 bytes of the key generated using the DH algorithm is used to encrypt the first block of data. This encrypted data is then used as a key for the second block. This process is repeated until the last block is encrypted. The key generated once will be valid only for that particular time window and whenever the key time expires the SMM will again launch the DH key protocol.

4.3.2.1 DH system

The communication between the client and cloud services is encrypted using Diffie-Hellman (DH) key generation algorithm [108] as shown in Figure 21. Diffie-Hellman is a key exchange algorithm based on modulo arithmetic that can be used to securely exchange keys between two systems that do not share any mutual keys. The two systems create a shared secret over an insecure communications channel. Simply transmitting a symmetric cipher key is clearly inadequate because anyone reading the traffic could use the key to decrypt anything encoded with it. After agreeing on a large modulus (m) and a common base number (x), each side picks a random number (its local secret); computes x to that power (mod m), and transmits this. Upon receiving this number, the other side raises it to its local secret power, and computes x to the product of the two powers. Anyone snooping on the connection can see the two partial powers transmitted, but without the secret powers the sniffer cannot compute the shared secret.

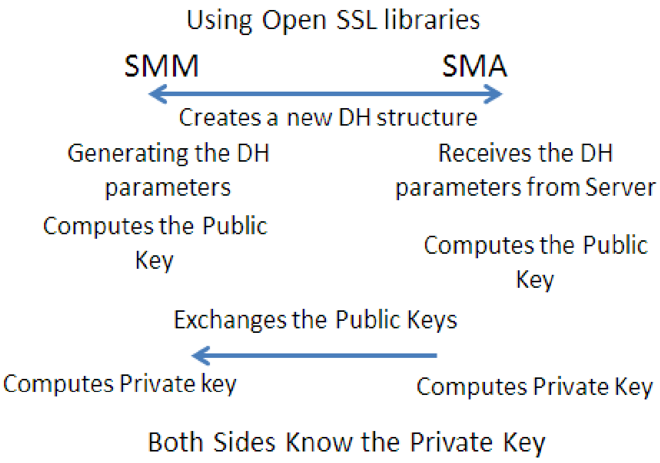
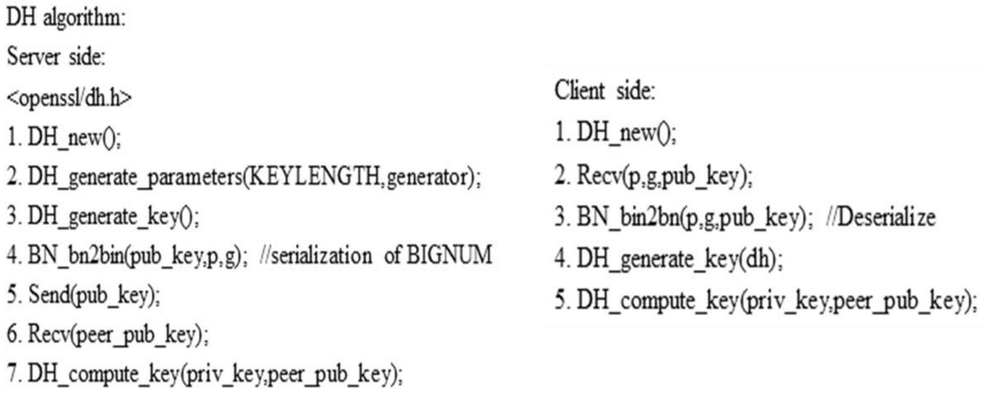

Using Open SSL libraries
SMM
SMA
Creates a new DH structure
Generating the DH parameters
Receives the DH parameters from Server
Computes the Public Key
Computes the Public Key
Exchanges the Public Keys
Computes Private key
Computes Private Key
Both Sides Know the Private Key

Figure 20: Diffie-Hellman Key exchange protocol

The master key generated by the SMM is sent to client through SSL session (see step 2-3 in Figure 22). The client then uses the key for DES encryption. This shared secret is converted to the DES key and is used for further communication between the client and the server. In the worst case scenario, if the key is compromised, it will be effective only during that time window because in the next time window the key will be different. We implemented DH system in C language and OpenSSL library.



```
DH algorithm:
Server side:
<openssl/dh.h>
1. DH_new();
2. DH_generate_parameters(KEYLENGTH,generator);
3. DH_generate_key();
4. BN_bn2bin(pub_key,p,g); //serialization of BIGNUM
5. Send(pub_key);
6. Recv(peer_pub_key);
7. DH_compute_key(priv_key,peer_pub_key);
```

```
Client side:
1. DH_new();
2. Recv(p,g,pub_key);
3. BN_bin2bn(p,g,pub_key); //Deserialize
4. DH_generate_key(dh);
5. DH_compute_key(priv_key,peer_pub_key);
```

Figure 21: DH algorithm for key generation and exchange

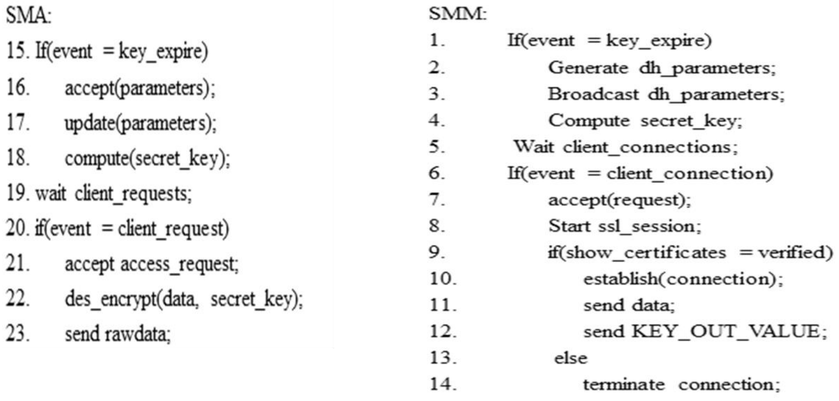
4.3.2.2 Key Hopping

Using the same key for a long time brings insecurity for the case if it is stolen and also, it requires a long computation time to encrypt file parts, incurring high overhead. To overcome this problem, we use shorter keys to reduce the time it takes to encrypt data and we change the keys randomly to increase the security of the storage service. The Storage Management Module keeps track of the time window and triggers the client and the server at the starting of the time window and when the time window ends. Thus the client and server follow the time window provided by the SMM. Any abnormal behavior by the client and server is monitored and responded to by the observer controller in the SMM. Once the time window ends, the keys that are used during that period will expire and SMM will initiate the generation of keys and then distribute them to various SMA. According to [113], it takes 73 days for a single Dual-core PC to crack a single RSA 512 bit key. However, through parallel processing, this time can be reduced significantly. Let us assume that the attacker is using 100 computers to crack a password which requires approximately 1 day. Thus, a 512 bit RSA key is unsecure if it is used for a long time period. In our evaluation, we used a time window of 4.8 hours which is very less time for an attacker to crack the key if one considers the other random variables used in our approach (window interval and file partitions). Our experimental results validated the resilience of our storage services against a wide range of attacks.

4.3.2.3 Client key Distribution

We use OpenSSL (Secure Socket Layer) [114] for the communication between the SMM and the client. The SMM certificates are verified on the client side to make sure that the key is received from the right sender. Once the certificate is verified, a secure socket is created for further communication. When the SSL session is established, the key is encrypted using MD5 ciphers [115]. A private and public key pair is generated and the public key is announced to the SMM. But the party which has the private key can only decode the entire key. After creating this secure channel the client obtains the key to decrypt the data. This key is then used to prove the clients identity to the SM module. After a successful authentication, the client will be added to the access control list. In further communication, if the client fails to prove its authenticity, the SMM will block its connection to the cloud storage system until the problem is resolved.

The algorithm to implement the RCSS is shown in Figure 23. The variables mentioned in the algorithm are self-explanatory. Initially all the timers are set and SMM launches the DH key generator between SMM and SMA (see step 1-5, Figure 23). Once the key is generated both the subsystems will wait for client connections (see step 6-9, Figure 23). The client request first goes to the SM module. The SMM will then share the computed key and its key time window (see step 1-5 Figure 23).



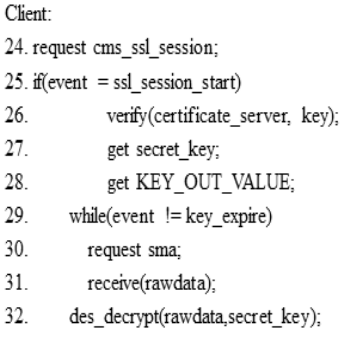
```
SMA:
15. If(event  = key_expire)
16.      accept(parameters);
17.      update(parameters);
18.      compute(secret_key);
19. wait client_requests;
20. if(event  = client_request)
21.      accept access_request;
22.      des_encrypt(data,  secret_key);
23.      send rawdata;
```

```
SMM:
1.        If(event  = key_expire)
2.             Generate dh_parameters;
3.             Broadcast dh_parameters;
4.             Compute secret_key;
5.         Wait client_connections;
6.        If(event  = client_connection)
7.             accept(request);
8.             Start ssl_session;
9.             if(show_certificates  = verified)
10.                 establish(connection);
11.                 send data;
12.                 send KEY_OUT_VALUE;
13.            else
14.                 terminate  connection;
```

Figure 22: Resilient Cloud Storage Service Algorithm

File Partitioning:

In May 2011, a popular file sharing service Dropbox was accused in a complaint to the Federal Trade Commission of using "a single encryption key for all the user data the company stores". The concern is that if a hacker was able to break into Dropbox's servers and obtain the key, it could gain access to all of the Dropbox's user data [116]. So to improve the resilience of stored data, it is important to partition data into several parts and use different keys for each data partition. This increases the overhead but it adds one more layer of security that attackers must overcome within a short period of time to succeed in accessing the data. Figure 24 shows the algorithm for encrypting the file by partitioning it and encrypting each part with a different key.



```
Client:
24. request cms_ssl_session;
25. if(event  = ssl_session_start)
26.             verify(certificate_server,  key);
27.             get secret_key;
28.             get KEY_OUT_VALUE;
29.       while(event  != key_expire)
30.          request sma;
31.          receive(rawdata);
32.       des_decrypt(rawdata,secret_key);
```

Figure 23: Algorithm for file partitioning

4.3.3 Evaluation and Results

In this section we present our implementation testbed and the performance results we obtained using RCSS architecture.

4.3.3.1 Testbed Configuration

In our experimental evaluation testbed, the storage servers are implemented using a cluster of virtual machines running on different nodes of an IBM Bladecenter HS22 Private Cloud [90]. Storage server 1 is implemented using Ubuntu 10.04 Linux operating system that runs on all its virtual machines that use the Hadoop Distributed File system. OpenSSL is used for establishing secure communication channel between SMA and storage systems. The SSL server provides the CA (Certificate Authority) certificate and Server certificate. Similarly the client system providers the CA certificate and client certificate. Whenever a client is requesting to connect to a server, the client certificate which is already signed by that server is verified by the CA and if the validation passes the communication is established.

4.3.3.2 Experimental Results and Evaluation

To evaluate the performance gain that can be achieved from using the key hopping technique, let us assume that there are 2000 sessions that use the resilient storage service. In our testbed, the average time to run an SSL processing time with 1024 bit key is approximately around 4 seconds. To run the RDS algorithm, the execution times of its components, if we assume the key size is 1024 and number of hops 5, are as follows: DH Protocol time is 11 seconds, key distribution time is 2 seconds, and DES encryption-/decryption is 1 second.

We use Performance Improvement Factor (PIF) as a metric to quantify the expected performance using our approach. The PIF metric can be computed as:

$$PIF = \frac{RT_{ssl} - RT_{RCS}}{RT_{ssl}}$$

RT_{ssl} = No of sessions * Time taken for SSl protocol

RT_{RCSS} = (T_{DH} protocol + $T_{keydistribution}$)*No of hops + (No of sessions * Time for $DES_{en+decryption}$)

where,

RT_{ssl} is the response time for system only with SSL.

RT_{RCSS} is the response time for the system with RCSS implementation.

T_{DH} protocol is time of execution for DH protocol.

$T_{keydistribution}$ is the time taken for client key distribution.

Based on our assumption for the number of sessions, the performance improvement (PIF) is 74%.

Figure 25 shows the overall overhead time for RCSS with respect to the number of hops. It is clear that as the number of hops increases the overhead increases.

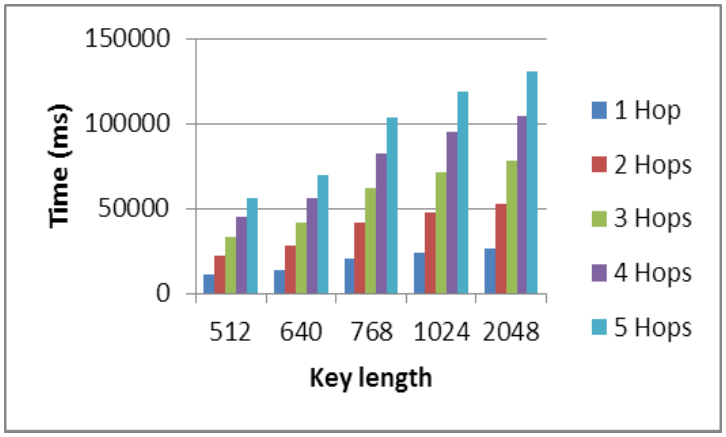


Figure 24: Performance overhead versus key length and number of hops

We also quantify the performance gain that can be achieved in encrypting different file sizes as shown in Figure 26. We compared the performance of using static key of size 2048 bits versus using a key hoping technique with 512 bits with two hops. The performance improvement factor is calculated as follows using the above mentioned assumptions.

$$PIF_{256MB} = \frac{RT_{2048ssl} - RT_{512ssl}}{RT_{2048ssl}}$$

where,

$RT_{2048ssl}$ = Response time with 2048 bit key and no hops.

RT_{512ssl} = Response time with 512 bit key, 2 hops using RDS approach.

The performance improvement factors for file sizes 256 MB, 64 MB, and 1 MB are 65.5%, 73.9% and 73.01% ,respectively.

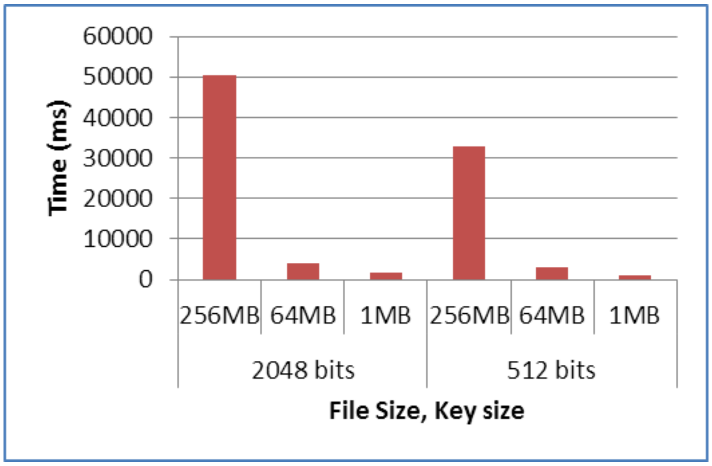


Figure 25: File size vs overhead time for different keys

Figure 27 shows how the overhead increases as we increase the number of parts for file partitioning. With a 512 bit long key and 2 hops, the overhead is almost 50% less than the overhead when we use 2048 bit long key with no hopping. This shows that by using smaller keys, we can improve the performance (50% with 2 hops as shown in Figure 27) while improving the security of the system by adding three security layers; for the attacker to succeed, the attacker needs to know the number of partitions, the keys used in each interval, and the length of each time window.

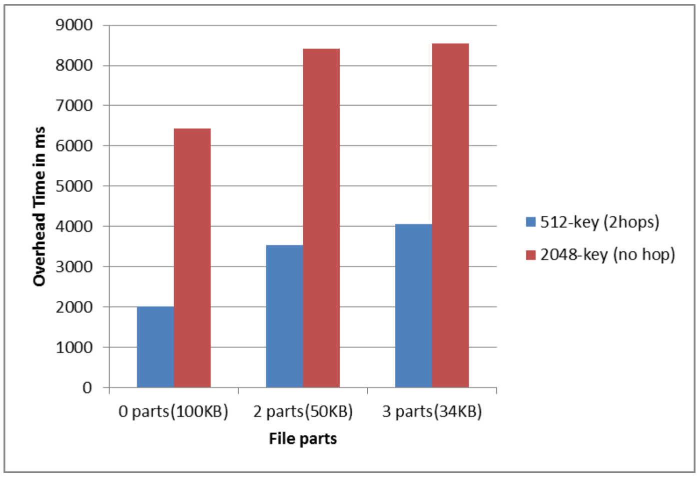


Figure 26: File parts vs overhead time for different keys

4.4 Resilience Modeling and Analysis

The process of quantifying resilience is a difficult process due to the heterogeneity of the environment, so a general and quantitative set of metrics for the resilience of cyber systems is accepted to be impractical [7]. Therefore, instead, we provide a method for quantifying resilience in an environment that is running our resilience approach. The method we propose to quantify the resilience of a cloud environment uses four important metrics: confidentiality, integrity, availability, and exposure. These metrics together can be used to represent the information security attributes for any system.

The attack surface of a software system is an indicator of the system's security; i.e., the higher the attack surface for a system, the lower the security is [117]. The attack surface represents the area in which adversaries can exploit or attack the system through attack vectors. In a SBE enabled environment, the attack surface measurement can be used to quantify the resilience. We will show that using SBE algorithm will decrease the attack surface, and therefore, increase the resilience compared to a static execution environment. The first step in quantifying the attack surface is identifying the metric for the software system; this includes the operating systems, programming languages, and the network. There are many tools that can be used to identify attack vectors, such as Microsoft Attack Surface Analyzer [118], Flawfinder [119], Nessus [120], Retina [121], and CVEChecker [107]. The application will always have an attack surface less than or equal to the system attack surface because the application, while it is running, will have a subset of the system attack surface; not all of the system attack vectors will apply to the application execution environment.

CVE (Common Vulnerabilities and Exposures) [107], which is a public reference for information security vulnerability and exposures, is used to determine the confidentiality, integrity, and availability of the software system. CVSS (Common Vulnerability Scoring System) [75] is used as a standard measurement system for industries, organizations, and governments that need accurate and consistent vulnerability impact scores. Cyber resilience depends on maintainability, dependability, safety, reliability, performability, and survivability which are all functions of Confidentiality, Integrity, and Availability [7]. Hence, we define the resilience as follows:

Definition: The system resilience R is the ability of the system to continue providing its Quality of Service (QoS) as long as the *impact* of the attacks is below the minimum threshold R.

The impact $i_v(t)$ of a vulnerability v is:

$$i_v(t) = \begin{cases} 0, t < T_v \\ I_V, t \geq T_v \end{cases}$$

Where T_v is the time required for discovering the vulnerability and exploiting it, and I_v is the impact of exploiting the vulnerability.

The expected value of the impact of a vulnerability v is given by:

$$E[i_v] = I_v \cdot Pr(A_v)$$

where A_v is the random variable that represents the occurrence of an attack exploiting vulnerability v. We can evaluate the probability of A_v as:

$$Pr(A_v) = Pr(A) \cdot Pr(U_v)$$

where A denotes the existence of an attacker who is trying to exploit the system and U_v denotes the time needed to successfully exploit the vulnerability v. To simplify the problem, we will assume that any attacker that spends more than T_v time in exploiting vulnerability v is successful, i.e., assume that all attackers are expert attackers and can successfully launch the attack in a minimum time T_v. By using the application life cycle time T_f and assuming that U_v is a uniform random variable, the pdf (probability density function) for U_v is given by:

$$Pr(U_v) = \begin{cases} 0, t < T_v \; or t > T_f \\ \dfrac{1}{T_f - T_v}, t \geq T_v \end{cases}$$

We define the impact of a system with N vulnerabilities to be:

$$i_{system} = E[i_{v_1} + i_{v_2} + \cdots + i_{v_N}]$$

Using the linearity property of the expected value, the previous equation can be re-written as:

$$i_{system} = E[i_{v1}] + E[i_{v2}] + \cdots + E[i_{vN}]$$

$$= I_{v_1} \cdot Pr(A) \cdot Pr(U_{v_1}) + I_{v_2} \cdot Pr(A) \cdot Pr(U_{v_2}) + \cdots + I_{v_N} \cdot Pr(A) \cdot Pr(U_{v_N})$$

$$= \sum_{k=1}^{N} I_{v_k} \cdot Pr(A) \cdot Pr(U_{v_k}) = Pr(A) \cdot \sum_{k=1}^{N} I_{v_k} \cdot Pr(U_{v_k})$$

Since we do not have a direct control over the $\Pr(A)$ or the impact value I_{vk} of the k-th vulnerability v_k, in our SBE technique, we continuously force the time t to be less than T_v for all or most vulnerabilities, which in turn forces $\Pr(U_v)$ for those vulnerabilities to always be zero. We are currently using the CVEChecker tool to get the impact score I_{v_k}.

Using the multiple functionally equivalent variants to run the application will significantly improve its resilience against attacks because of the reduced successful attack probability on the application execution environment. For example, by using L functionally equivalent versions of the application, the probability of successfully exploiting an existing vulnerability v_k is given by:

$$Pr(U_{v_k}) = Pr(U_{v_k,1} \cap U_{v_k,2} \cap \ldots \cap U_{v_k,L})$$

Since these versions are independent from one another:

$$Pr(U_{v_k}) = Pr(U_{v_k,1}) \cdot P\,r(U_{v_k,2}) \cdot \ldots \cdot P\,r(U_{v_k,L})$$

Assuming that all versions are equally likely to be attacked:

$$Pr(U_{v_k}) = \frac{1}{L} Pr(U_{v_k}) \cdot \frac{1}{L} Pr(U_{v_k}) \cdot \ldots \cdot \frac{1}{L} P\,r(U_{v_k}) = \left(\frac{1}{L} Pr(U_{v_k})\right)^{L}$$

Figure 28 shows the decrease in the probability of a successful attack as a function of the number of versions to be used in the SBE algorithm.

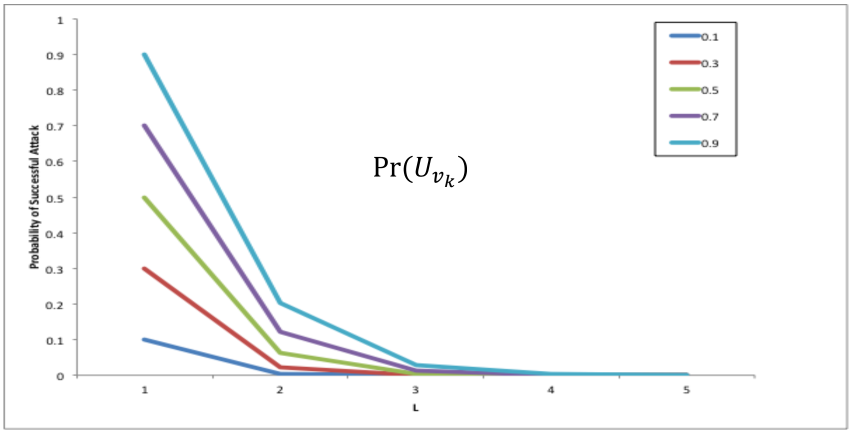


Figure 27: Probability of Successful Attack with respect to the number of versions

From the previous discussion, it is clear that our SBE technique will significantly reduce the ability of attackers to exploit existing vulnerabilities in cloud applications.

4.5 Resilient Cloud Services (RCS)

For the cloud to be fully adopted and effectively used it is critical that the security mechanisms are robust and resilient to faults and attacks. Securing cloud applications and services is a challenging research problem because it involves many interdependent tasks including vulnerability scanning, application layer firewalls, configuration management, alert monitoring and analysis, source code analysis, and user identity management. Current security techniques are mainly signature based and manual intensive. As a result, it is widely believed that software systems and network protocols will always have vulnerabilities that can be exploited by cyberattacks. Furthermore, the monoculture problem of software makes vulnerability exploitations propagate instantly to a large number of computers and network devices. In this section, we present an environment to develop resilient cloud services (RCS) and evaluate their performance.

4.5.1 RCS Development Environment

We achieve resilient operation by continuously hiding the execution cloud systems using two runtime algorithms: cloud SBE that was discussed before and autonomic management. The RCS methodology makes it extremely difficult for an attack to disrupt the normal operations of a cloud application. Also, the dynamic change in the execution environment hides the software flaws that would otherwise be exploited by a cyberattacker. The autonomic management algorithm is needed to provide the dynamic configuration capabilities to hide the cloud execution environment at runtime. Figure 29 illustrates the RCS development environment, which includes the following main modules: Application Resilient Editor (ARE), Cloud Resilient Middleware (CRM), Supervisor VMs (SVMs), and Master VMs (MVMs). In what follows, we briefly highlight the main functions to be provided by each module.

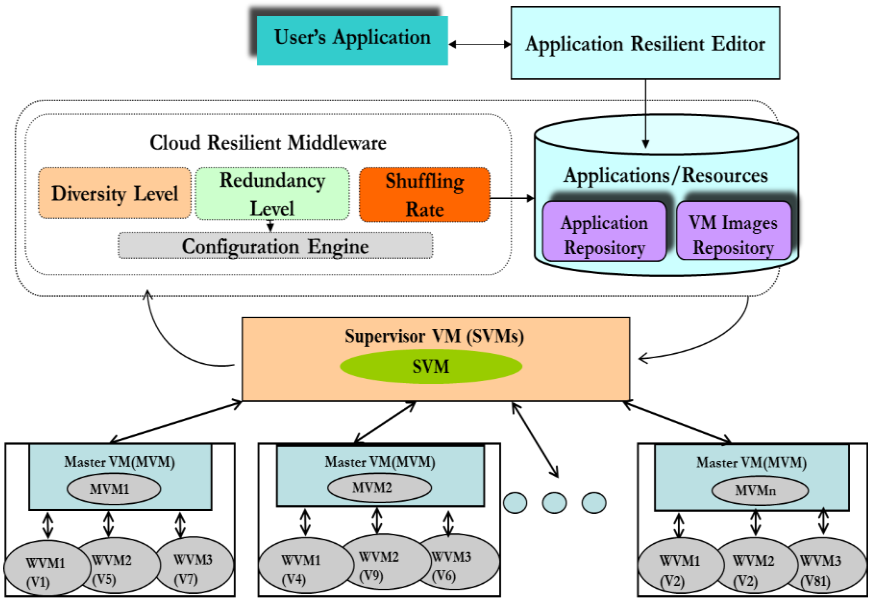


Figure 28: RCS Development Environment.

4.5.1.1 Application Resilient Editor

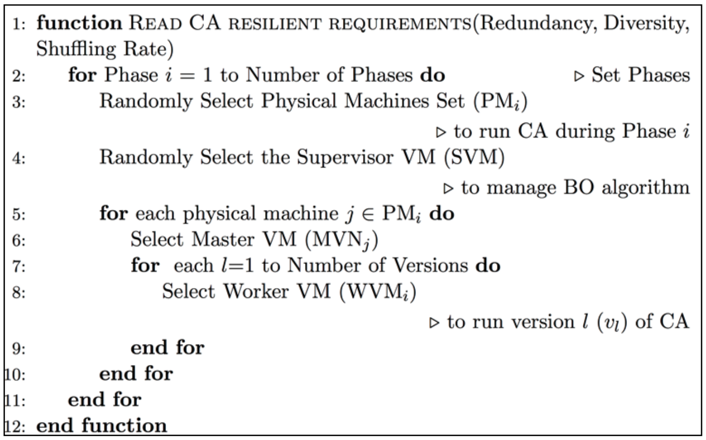
The editor allows users and/or cloud application developers to specify the resilient requirements of the cloud applications. The resilient requirement can be characterized by: 1) defining the required diversity level (how many different versions of an application and/or how many different platforms (e.g., operating system types) that are required to run the application; 2) defining the redundancy level (how many redundant physical machines are required); and 3) defining how often the execution environment needs to be changed and the number of application execution phases.

4.5.1.2 Cloud Resilient Middleware (CRM)

The CRM provides the control and management services to deploy and configure the software and hardware resources that are required to achieve the application resilient requirements as specified by the editor. The resilient operation for any cloud application is achieved using the SBW algorithm that hides the execution environment by dynamically changing the number of versions used to run the application at each phase. The decisions regarding when to shuffle the current variant, the shuffling frequency, and the variant selection for the next shuffle are guided by a continuous monitoring and analysis of current execution state of cloud applications and the desired resilience requirements.

To speedup the process of selecting the appropriate resilient algorithms and execution environments, the CRM repository contains a set of SBE algorithms and images of virtual machines that run in different operating systems (e.g., Windows, Linux, etc.) to implement supported cloud applications and services such as Map/Reduce, Web services, Request and Tracker (RT) applications, just to name a few.

The Configuration Engine (CE) takes the resilient requirements specified by the users using the CRM editor and uses the CRM repository to build the execution environment that achieves the required resilient cloud operations or services. The selected SBE algorithm runs each Cloud Application (CA) as a sequence of execution phases, where each phase is administered by one Supervisor Virtual Machine (SVM). The SVM manages several Master Virtual Machine (MVMs) each of which run on different physical machines in order to tolerate attacks that might discover the physical machine running the CA during one phase. Furthermore, each MVM manages the voting algorithm on the results produced by several Worker Virtual Machines (WVMs) where each WVM runs different version of the cloud application. The CE algorithm is shown in Figure 30. To explain the CE algorithm, we use the example shown in Figure 31 as a running example. The number of phases in this CA example is two (step2), and the supervisor virtual Machine for Phase 1 is SVM$_2$ and SVM$_3$ for Phase 2 (step 4). In step 6, we select the masters for each phase (MVM$_1$, MVM$_{2,}$ MVM$_3$). Similarly, we create three Worker VMs (WVMs) to run each version of the CA application during each phase (Step 7). In this example, during phase 1, master virtual machine MVM1 will be managing the parallel execution of three versions (V1, V4, and V3) on three worker virtual machines, while MVM$_2$ manages the parallel execution of V8, V2, and V5, and MVM3 manages another set of three versions of the CA application (V9, V2, and V3). The supervisor virtual machine (SVM$_1$) of Phase I will collect the results from the three masters and pass the output produced by the voting algorithm as will be explained later. Similar steps are followed during the second phase as shown in Figure 31.



```
1: function READ CA RESILIENT REQUIREMENTS(Redundancy, Diversity,
   Shuffling Rate)
2:     for Phase $i = 1$ to Number of Phases do                ▷ Set Phases
3:         Randomly Select Physical Machines Set (PM$_i$)
                                          ▷ to run CA during Phase $i$
4:         Randomly Select the Supervisor VM (SVM)
                                          ▷ to manage BO algorithm
5:         for each physical machine $j \in$ PM$_i$ do
6:             Select Master VM (MVN$_j$)
7:             for  each $l$=1 to Number of Versions do
8:                 Select Worker VM (WVM$_i$)
                                          ▷ to run version $l$ ($v_l$) of CA
9:             end for
10:        end for
11:    end for
12: end function
```

Figure 29: Configuration Engine Algorithm

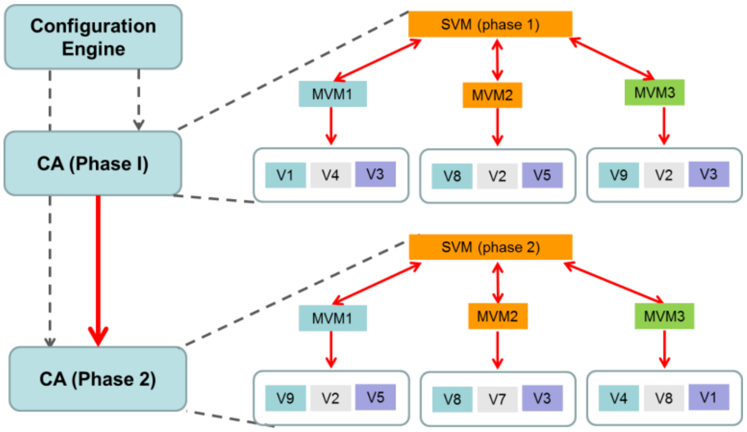
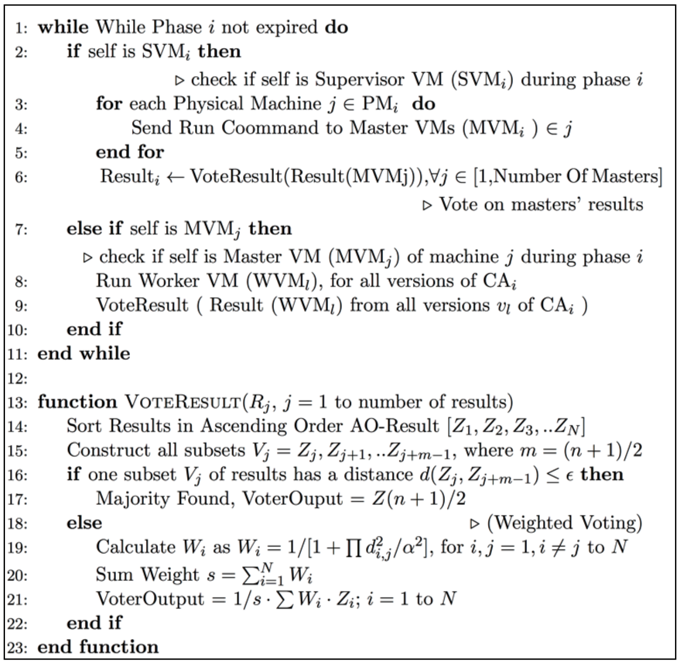


Figure 30: SBE algorithm main components for two-phase example application

4.5.1.3 Software Behavior Encryption (SBE) Algorithm

Once the CE setups the environment, the selected supervisor virtual machine (SVM) for each phase will administer the SBE algorithm as shown in Figure 32. The SVM is designed to manage the resilient behavior obfuscation of the selected algorithm by the configuration engine as discussed before. The SVM$_i$ for Phase I manages the operations of all the VMs involved in computing the CA during its assigned phase as shown in Figure 32. The designated SVM will run the designated MVM at each physical machine used in a given phase (Step 4), and then collects the results from all the masters to choose the one to be passed to the next phase. The voting procedure (Step 13 through 23), which is based on an integration voting algorithm [76], first check results to see if majority vote can be achieved (Steps 12 to 17) when the difference between results is less than an acceptable threshold ε. If the difference in the results is larger than the acceptable threshold, a weighted voting procedure is used to determine the result to be passed to the next phase (Steps 19 through 22).



```
1:  while While Phase $i$ not expired do
2:      if self is SVM$_i$ then
                        ▷ check if self is Supervisor VM (SVM$_i$) during phase $i$
3:          for each Physical Machine $j \in$ PM$_i$ do
4:              Send Run Coommand to Master VMs (MVM$_i$ ) $\in j$
5:          end for
6:          Result$_i \leftarrow$ VoteResult(Result(MVMj)),$\forall j \in$ [1,Number Of Masters]
                                                        ▷ Vote on masters' results
7:      else if self is MVM$_j$ then
                ▷ check if self is Master VM (MVM$_j$) of machine $j$ during phase $i$
8:          Run Worker VM (WVM$_l$), for all versions of CA$_i$
9:          VoteResult ( Result (WVM$_l$) from all versions $v_l$ of CA$_i$ )
10:     end if
11: end while
12:
13: function VoteResult($R_j$, $j = 1$ to number of results)
14:     Sort Results in Ascending Order AO-Result $[Z_1, Z_2, Z_3, ..Z_N]$
15:     Construct all subsets $V_j = Z_j, Z_{j+1}, ..Z_{j+m-1}$, where $m = (n+1)/2$
16:     if one subset $V_j$ of results has a distance $d(Z_j, Z_{j+m-1}) \leq \epsilon$ then
17:         Majority Found, VoterOuput $= Z(n+1)/2$
18:     else                                                  ▷ (Weighted Voting)
19:         Calculate $W_i$ as $W_i = 1/[1 + \prod d^2_{i,j}/\alpha^2]$, for $i, j = 1, i \neq j$ to $N$
20:         Sum Weight $s = \sum_{i=1}^{N} W_i$
21:         VoterOutput $= 1/s \cdot \sum W_i \cdot Z_i$; $i = 1$ to $N$
22:     end if
23: end function
```

Figure 31: SBE Algorithm for one phase (CA, Phase I)

4.5.2 RCS Experimental Results and Evaluation

In our evaluation of the RCS approach, we used the MapReduce cloud service as a running example to evaluate the effectiveness and performance of our method to achieve resilient cloud applications. MapReduce [85] is being widely used as a powerful parallel data processing programming model to solve a wide range of large-scale computing problems. With the MapReduce programming model, programmers need to specify two functions: *Map* and *Reduce*. The Map function receives a key/value pair as input and generates intermediate key/value pairs to be further processed. The Reduce function merges all the intermediate key/value pairs associated with the same (intermediate) key and then generates final output. There are three main roles: the master, mappers, and reducers. The single master acts as the coordinator responsible for task scheduling, job management, etc. MapReduce is built upon a distributed file system (DFS) that provides distributed storage. The input data is split into a set of *M* blocks, which will be read by *M* mappers through DFS I/O. Each mapper will process the data by parsing the key/value pair and then generate the intermediate result that is stored in its local file system. The intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together (the shuffle phase). The locations of the intermediate results will be sent to the master who notifies the reducers to prepare to receive the intermediate results as their input. Reducers then use Remote Procedure Call (RPC) to read data from mappers. The user defined reduce function is then applied to the sorted data; basically, key pairs with the same key will be reduced in some way, depending on the user defined reduce function. Finally the output will be written to DFS. For example, Hadoop [86] is an open source implementation of the MapReduce framework and MRS-MapReduce [77] is another implementation.

The RCS testbed consists of several compute nodes and controller as shown in Figure 33. The reported results were generated using three Dell XPS 8700 towers with i7 4770 processors and 12GB memory, with Ubuntu 12.04 Server as a host operating system. We deployed OpenStack Havana to create a private cloud computing environment. OpenStack is a free and open-source cloud management software "to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable." [81]. For a cloud environment, mainly OpenStack consists of a controller node (that has the full control over the environment), at least one compute node (to run VMs on a hypervisor – we have chosen KVM as our hypervisor since it is one of the native hypervisors OpenStack is supporting), and a network node (for managing networks), combined or separately, with services such as Keystone (identity manager), Glance (image service), etc.

In our implementation, we have used one of the physical machines (PM) as a controller with the compute capabilities (i.e. it can control all the VMs in the cloud environment and can also spawn VMs) and with nova-network (to manage the networking). Our RCS environment can handle hundreds or thousands of machines. In each phase, the system selects a random number of physical machines, $n \epsilon N$ physical machines, is chosen to run the required VMs in order to provide the resilient cloud services. Figure 33 shows the topology of the testbed to experiment with and evaluate the RCS performance. An internal network switch is used for the service operations to be able to communicate with each other and an external router has been used with special Access Control List (ACL). The ACL has been updated in a way that no access from outside to the system or no access from the VMs to the outside is possible; but only to the Web Interface is possible. This step has been *implemented* to introduce additional security to block the intrusions and to prevent the data leakage to the outside world. On the controller node we allocate a Web interface VM (Web iface) that can be used to submit jobs and/or to use application resilient editor explained earlier. In addition various types of VM images (with the required programs installed on them) are stored in Glance service so they can be spawned as required by the configuration engine.

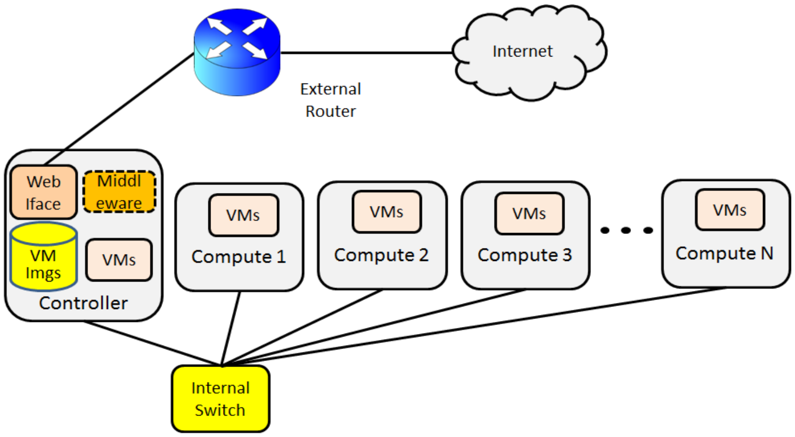


Figure 32: Testbed topology.

In order to experiment and evaluate the RCS performance and their effectiveness to tolerate any type of malicious attacks against the MapReduce cloud service, we have used WordCount application which is a typical MapReduce application that counts the occurrences of each word in large input text data. We have created two versions of this application by using two different MapReduce platforms such as Hadoop and MapReduce. We have also used different programming languages (Java and Python) for Hadoop environment in order to generate different versions (diversified execution environments) of this application.

Table 2 shows the execution time of all the environments for a single file (with different sizes) along with the overhead of the ARCM approach. During the experiments, the files with the specified sizes are created by merging the largest novels in English literature (such as Romeo and Juliet from William Shakespeare). In our experiments, the MapReduce implementations first apply Map and then apply Map and Reduce for all the Map outputs. The reason behind this technique is that when multiple files are used for MapReduce, there is a need to a Map operation for the combined Map outputs since the Reduce function requires inputs to be sorted.

In addition to evaluate the overhead of our resilient approach, we also evaluated the evaluated the effectiveness of our ARCM approach against attacks and its ability to continue to operate normally in spite of these attacks. We apply Hydra and HPing3 in order to attack the systems while they are operating. We have applied the attacks when the systems start operating and evaluated their operations under attacks.

Table 2: The execution time of all the environments for a single file

	Execution Time (sec)		
File Size (MB)	NO_BO	With BO	Overhead (%)
5	108	100	7
10	118	107	10
15	130	114	13
20	135	122	9
25	152	130	14
30	164	138	16
60	235	182	23
138	314	247	21

Hydra: Hydra is a brute force password cracking software used to guess the password by using a database or by trying all the combinations. An example of an attack using Hydra would be as follows:

```
hydra 192.168.1.22 ssh2 -s 22 -P pass.txt -L users.txt -e ns -t 10
```

In this example, Hdyra is used to apply attack on the 192.168.1.26 remote computer over the SSH on port 22 using the pass.txt password database file and users.txt users file. It creates 10 threads to check all the possibilities to have a faster result since it takes a while to check all the usernames and passwords combinations. In our experiments, we use hydra to try to find the password of the victim VM. Since it checks all the combinations using SSH protocol, the victim VM needs to reply and hence, a high amount of resource utilization is observed. Therefore, hydra also create a DoS attack behavior.

Hping3: Hping3 is a networking tool used to create and send custom TCP/IP packets used for security auditing and network/firewall testing purposes. By sending large amount of synchronization packets with large sizes to a specific port, it is possible to apply DoS/DDoS attacks. It also allows using random IP source by spoofing IP addresses which helps the attacker to hide his IP address. Using this approach, we have executed a DoS attack from an attacker VM to the target VM and made it unable to respond to any SSH connections. Since it also creates high resource utilization on the victim VM, the operations required to run MapReduce either took longer time or could not complete the execution.

In Figure 34 we show how much the execution time would change when there is an attack for both systems and in Figure 35 we show how much additional overhead is observed during the attacks. In these experiments, we applied Hydra and HPing3 to the execution environment as DDoS attack by deploying the tools on multiple attack VMs. The results demonstrate that only a small increase in the overhead is seen for the resilient architecture (RCS) and for the non-resilient case (NO-RCS), the executions have crashed due to DDoS. In these experiments, we have used a large file (138MB) that was created by merging the largest novels multiple sizes.

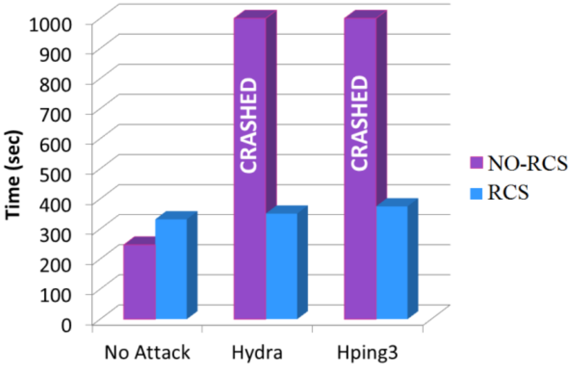


Figure 33: Execution time of the resilient and no resilient architectures with Hydra and Hping3

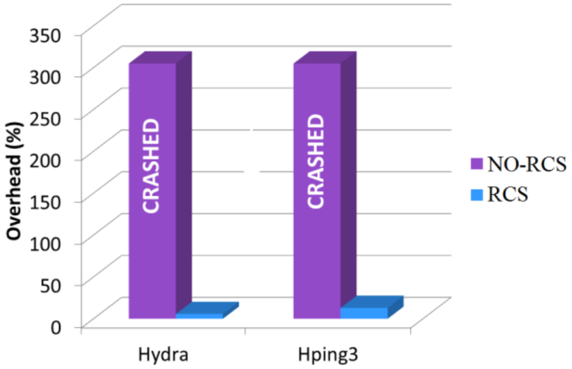


Figure 34: Hping3 with compared to the no attack scenarios

In what follows, we describe how the SBE algorithm can tolerate a wide range of cyberattacks that were launched against the MapReduce application.

4.5.2.1 Attack Analysis and Evaluation

In our evaluation, we considered two types of malicious attacks against the MapReduce application: Denial of Service (DOS) and insider attacks. Below is the list of the attacks used in our evaluation.

Denial of Service (DoS) Attack: Both CPU and memory of the infected system are consumed by the DoS attack. The MapReduce implementations cannot operate under such condition due to the lack of available resources. The RCS method uses the redundant WVMs (three WVMs on each physical machine) and, hence, the results will be obtained from the other WVMs that have not been hit by this attack. It is also important to note that these WVMs and MVMs are randomly selected during each phase of the computations, so it will be extremely difficult for the DoS attack to indetify and affect the active MVMs and WVMs during a given phase, in a cloud environment. But, as discussed before, if the DoS succeeds, the redundant WVMs on other physical machines will be able to provide the correct results to the next phase according to the BO algorithm.

Fork Bombing Attack: In this scenario, the attacker applies techniques like fork bombing to stop the usability of the VMs. When a WVM is infected, the redundant WVMs will be able to provide the correct output. The infected WVM, on the other hand, will be restarted and a new SSH keypair set will be assigned and consequently, the attacker will lose its connection to the system.

Change of Authorized Keys: In such a scenario, the attacker changes the authorized keys; however, this will be overcome by injecting new keys at every phase. So the infected VM is only inaccessible by the users for one phase. Also, the other redundant WVMs that are not affected by this compromise will be able to deliver the error free output the MVM and later to the SVM.

Insider Threat Attack: The insider attacker in this case can force the system to change the output results so the user ends up receiving incorrect results. However using the voting algorithm on multiple WVMs on each physical machine will allow us to detect and mask the injected error. Furthermore, the infected WVM will be rebooted and the key-pairs will be changed to make the attacker lose the connection.

Insider Attack that Arbitrary Shut-Down Services: The MapReduce services are initialized in the beginning of the system and an attacker can try to cause a physical machine not to operate by shutting down the services. In such a case, the infected machine will have a new keypair set and will be re-initialized in the beginning of the next phase. However, the other redundant physical machines will be able to tolerate such an attack. Furthermore, the compromised physical machine will be cleared so it can continue its normal operation in the second phase.

User to Root (U2R): This type of attacks uses login access mechanisms to bypass normal authentication and thus gain the privileges of another user (e.g., root privileges). Since the execution of the VMs are limited job and also voting mechanism is used to detect abnormal behavior in the workers, such an attack will not be affecting the final output of the system.

Remote to Local (R2L): An unauthorized remote user tries to bypass normal authentication and execute commands and programs on the target machine as any authorized local user. Similar to the U2R attacks, this attacks will not be affecting the system behavior and final result.

Probe: Potential target resources are tested to gather information. These are usually harmless (and common) unless vulnerability is discovered and later exploited. With the dynamic behavior of the systems using diversity and MTD, the potential vulnerabilities will not be probed and exploited successfully.

Worm/virus: Malicious program/code that can propagate either by itself or by user activities and result in widespread impact on large network (e.g., Internet). By having diversity in the platforms, the successful worms/viruses will not be able to affect the whole execution environment.

5. Conclusions and Research Contributions

While DDDAS-based computing is emerging as a promising paradigm, security is a significant barrier to its adoption. In this report, we first presented an overview of the current security issues in cyber systems and cloud computing. We summarized previous works that classified cloud security issues on the basis of cloud delivery models and the components of the cloud. Further, we also observed that attacks on cyber systems such as DDDAS environments cannot be prevented. In order to overcome this challenge, we developed a design methodology to develop resilient DDDAS environment (rDDDAS) that is based on the following capabilities: Redundancy, Diversity, Shuffling, and Autonomic Management. In the rDDDAS methodology, we adopt diversity technique to the DDDAS execution environment, redundancy in the resources used to run the DDDAS services and randomly changing the versions and resources used to make it prohibitively expensive for attackers to figure the current cloud service execution environment and succeeding in exploiting vulnerabilities and launching attacks. We also presented a testbed to validate the rDDDAS environment and its resilient algorithms using three applications (MapReduce, Jacobii's iterative linear equation solver, and some programs from the MiBench benchmark suite). Our experimental results showed that our resilient cyber services can tolerate a wide range of attack scenarios with around 7% of overhead time. As a future research direction, we are currently working on developing analytics techniques to quantify the resilience of different rDDDAS implementation strategies, overhead and performance of the rDDDAS services.

A summary of the main contributions of our research can be highlighted in the following points:

1) **Developed a methodology to build resilient DDDAS (rDDDAS)** that utilizes Moving Target Defense (MTD) and Software Behavior Encryption (SBE) techniques to make it extremely difficult for attackers to exploit existing vulnerabilities or compromise DDDAS environments.

Benefits: By adding resilient capabilities to DDDAS paradigm, it will make DDDAS attractive to develop DOD mission critical applications including crisis and cyber battle management systems.

Publications:

- J. Pacheco, C. Tunc, and S. Hariri, "Design and Evaluation of Resilient Cyber Infrastructures for Smart Cities," to be presented at the IEEE Second International Smart Cities Conference (ISC2 2016), September 2016, Trento, Italy.
- G. Dsouza, G. Rodriguez, Y. Al-Nashif, and S. Hariri, "Building Resilient Cloud Services using DDDAS and Moving Target Defense," International Journal of Cloud Computing, Vol 2, No. 2/3, 2013, pp. 171-190.
- G. Dsouza, S. Hariri, Y. Al-Nashif, and G. Rodriguez, " Resilient Dynamic Data Driven Application Systems (rDDDAS)", Proceedings of International Conference on Computational Science, Barcelona, Spain, 5-7 June, 2013.
- G. Dsouza, H. Alipour, S. Hariri, Y. Al-Nashif, and M. Eltoweissy, "Cloud Resilient Architecture," in Proceedings of the 1st IBM Cloud Academy Conference (ICA CON 2012), Research Triangle Park, NC, April 19-20, 2012.

2) **Developed Resilient Cloud Services (RCS)** to validate the feasibility of the rDDDAS methodology to build a wide range of resilient DDDAS applications.

Benefits: DDDAS developers can use the rDDDAS middleware and software components such as Map/Reduce, Jacobi's Iterative Linear Equation Solver, and MibBench Benchmarks to develop large scale resilient DDDAS applications.

Publications:

- C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, J. Hughes, “Autonomic Resilient Cloud Management (ARCM) Design and Evaluation,” in 2014 IEEE International Conference on Cloud and Autonomic Computing (ICCAC), 2014 p. 44-49.
- C. Tunc, S. Hariri, and A. Battou, “A Design Methodology for Developing Resilient Cloud Services (RCS),” Handbook of System Safety and Security: Cyber Risk and Management, Cyber Security, Threat Analysis, Functional Safety, Software Systems, and Cyber Physical Systems. Edited by Edward Griffor, Elsevier Inc., to be published in 2016.

3) **Developed analytical models to quantify resilience in SBE applications** and showed that by using three diversified versions, we can tolerate with high probability any type of cyberattacks in spite of using vulnerable applications or software components.

Benefits: enables DDDAS decision support system to use analytical models to configure the SBE algorithms such that DDDAS environments can meet dynamically the required security and resilient requirements.

Publications:

- E. Blasch, Y. Al-Nashif, S. Hariri. “Static Versus Dynamic Data Information Fusion Analysis Using DDDAS for Cyber Security Trust. Procedia Computer Science 2014; 29:1299-1313.
- E. Blasch, Y. Badr, S. Hariri, and Y. Al-Nashif, “Fusion Trust Service Assessment for Crisis Management Environments,” pages 389-420, a chapter in a book “Fusion Methodologies in Crisis Management: Higher Level Fusion and Decision Making,” Editors: Galina Rogova, Peter Scott, Springer, ISBN 978-3-319-22527-2, (2016).
- Y. Badr, S. Hariri, Y. Al-Nashif, E. Blasch, “Resilient and Trustworthy Dynamic Data-Driven Application Systems (DDDAS) Services for Crisis Management Environments,” Procedia Computer Science, ICCS 2015 International Conference On Computational Science, Volume 51, 2015, Pages 1–15, 2015.

5.1 Broader Impacts and Technology Transfer

The results and tools developed in the DRCS project have been well received by DoD organizations and industry, and several projects have been funded to further develop and transition the technology to DoD market and industry. Below is a list of recent awards we received to deploy the rDDDAS capabilities into different DoD organizations:

- OSD SBIR Phase I Award (OSD153-005): Resilient Middleware Services for Cyber Physical Systems, Department of Army, Research, Development and Engineering Command. Jointly with AVIRTEK and UA, estimated starting date, August 2016.
- Army STTR Phase I Award (A16A-T010): Tactical Immune System (TIS), jointly with UA and AVIRTEK, estimated starting date: August 2016.
- NIST Award: Resilient Cloud Services (RCS): The goal of this project is to transition RCS capabilities to NIST Super Cloud infrastructure, estimated starting date: September 2016.
- US Army NETCOM: A prototype Resilient Cloud Services (RCS) environment has been developed for US Army Netcom based on VMware platform.
- NAVY Tactical Cloud: The Navy selected the RCS methodology developed in this project to apply it to their Tactical Cloud Services. The project could not start because of budget cut to the Navy tactical cloud program.

5.2 Students/Graduates

The project research supported partially many graduate students who received Ph.D. and M.Sc. degrees and are currently working in academia and industry. Below is the list of student names that were involved in the cybersecurity and resilience research tasks carried out in this project.

5.3 Ph.D. Students

1. Hamid Alipour – He received his Ph.D. degree from the ECE department of the University of Arizona in 2013. His research focused on cybersecurity and autonomic protection systems. He is currently working at Microsoft.

2. Farah Fargo – She received her Ph.D. degree from the ECE department of the University of Arizona in 2015. Her research focused on cloud computing and cloud security. She is currently working at Intel.

3. Cihan Tunc – He received his Ph.D. degree from the ECE department of the University of Arizona in 2015. His research areas focused on power/energy and performance management of the cloud computing systems, cloud security, and cloud systems for teaching. He is currently a Research Assistant Professor in the ACL lab at the ECE department.

5.4 M.Sc. Students

1. **Glynis Dsouza** – She received her M.Sc. degree from the ECE department in May 2013. Her research focused on software resilience, resilient computing, and cloud services. She is currently working at IBM.

2. **Hemayamini Kurra** – She received her M.Sc. degree from ECE department in May 2014. Her research focused on resilient storage systems and cloud security. She is currently working at IBM.

3. **Pratik Satam –** He received his M.Sc. degree from ECE department in 2015. His research focused on WiFi security, computer network security. He is currently a Ph.D. student in the ACL lab at the ECE department.

4. **Jin Bai** – He received his M.Sc. degree from ECE department in 2015. His research focused on anomaly based security for DNP3 protocol. He is currently working at Higgins Lab.

5. **Bilal Al-Baalbaki** – He received his M.Sc. degree from ECE department in 2015. His research focused onautonomic protection system focusing on ZigBee protocol. He is currently employed at General Motors.

6. **Navin Chaganti –** He received his M.Sc. degree from ECE department in 2015. His research focused on data analytics for behavior analysis. He is currently employed by KPMG.

7. **Nishant Prakash** – He received his M.Sc. degree from ECE department in 2015. His research focused on autonomous monitoring for insider threats. He is currently employed by Hewlett-Packard.

8. **Shrivatsa Upadhye** – He received his M.Sc. degree from ECE department in 2015. His research focused on cybersecurity lab as a cloud service. He is currently working in Netapp.

9. **Avinash Gudagi** – He received his M.Sc. degree from ECE department in 2015. His research focused on resilience quantification. Currently he is employed by Intel.

5.5 Selected other publications

1. H. Alipour, Y. Al-Nashif, P. Satam, and S. Hariri, “Wireless Anomaly Detection Based on IEEE 802.11 Behavior Analysis,” IEEE Transactions on Information Forensics and Security, Volume 10, Issue 10, pages 2158-2170, May 2015, ISSN:1556-6013.

2. H. Kholidy, F. Baiardi, and S. Hariri, “DDSGA: A Data-Driven Semi-Global Alignment Approach for Detecting Masquerade Attacks,” IEEE Transactions on Dependable and Secure Computing, Vol 12, No. 2, March/April 2015, pp 164-178.

3. D. Chen, L. Wang, A.Y. Zomaya, M.G. Dou, J. Chen, Z. Deng, and S. Hariri, “Paralell Simulation of Complex Evacuation Scenarios with Adaptive Agent Models,” IEEE Transactions on Parallel and Distributed Systems 26, no. 3 (2015): 847-857.

4. P. Satam, H. Alipour, Y. Al-Nashif, and S. Hariri, “Anomaly Behavior Analysis of DNS Protocol,” Journal of Internet Services and Information Security (JISIS), Volume 4, No. 3, November 2015.

5. D. Thebeau, B. Reidy, R. Valerdi, A. Gudagi, H. Kurra, Y. Al-Nashif, S. Hariri, and F. Sheldon, "Improving Cyber Resiliency of Cloud Application Services by Applying Software Behavior Encryption (SBE)." Procedia Computer Science 28 (2014): 62-70.

6. E. Blasch, Y. Al-Nashif, S. Hariri. “Static Versus Dynamic Data Information Fusion Analysis Using DDDAS for Cyber Security Trust. Procedia Computer Science 2014;29:1299--1313.

7. J. Bai, Y. Al-Nashif, and S. Hariri, “A Network Protection Framework for DNP3 Over TCP/IP Protocol, In:” Procedia IEEE AICCSA Conference, 2014.

8. Z. Pan, Y. Al-Nashif, and S. Hariri, “Anomaly Based Intrusion Detection for Building, Automation and Control Networks,” In Procedia IEEE AICCSA Conference, 2014.

9. C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, and J. Hughes, “Autonomic Resilient Cloud Management (ARCM) Design and Evaluation,” IEEE International Conference on Cloud and Autonomic Computing (ICCAC), 2014 p. 44-49.

10. F. Fargo, C. Tunc, Y. Al-Nashif, A. Akoglu, and S. Hariri, “Autonomic Workload and Resources Management of Cloud Computing Services,” IEEE International Conference on Cloud and Autonomic Computing (ICCAC), 2014,p. 101-110.

11. H. Kurra, Y. Al-Nashif, and S. Hariri, “Resilient Cloud Data Storage Services,” proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, Article No. 7,

12. D. Thebeau, B. Reidy, R. Valerdi, A. Gudagi, H. Kurra, Y. Al-Nashif, S. Hariri, F. Sheldon, “Improving Cyber Resiliency of cloud application services by applying software behavior encryption (SBE)”, 2014 Conference on Systems Engineering Research.

13. G. Dsouza, G. Rodriguez, Y. Al-Nashif, and S. Hariri, “Building Resilient Cloud Services using DDDAS and Moving Target Defnese,” International Journal of Cloud Computing, Vol 2, No. 2/3, 2013, pp. 171-190.

14. G. Dsouza, S. Hariri, Y. Al-Nashif, and G. Rodriguez, “Resilient Dynamic Data Driven Application Systems (rDDDAS)”, Proceedings of International Conference on Computational Science, Barcelona, Spain, 5-7 June, 2013.

15. G. Dsouza, H. Alipour, S. Hariri, Y. Al-Nashif, and M. Eltoweissy, “Cloud Resilient Architecture,” in Proceedings of the 1st IBM Cloud Academy Conference (ICA CON 2012), Research Triangle Park, NC, April 19-20, 2012.

6. References

[1] G. Dsouza, S. Hariri, Y. Al-Nashif, and G. Rodriguez, “Resilient Dynamic Data Driven Application Systems (rDDDAS)”, Proceedings of International Conference on Computational Science, Barcelona, Spain, 5-7 June, 2013.

[2] S. Hariri, C. Tunc, P. Satam, F. Al-Moualem, and E. Blasch. “DDDAS-Based Resilient Cyber Battle Management Services (D-RCBMS),” In 2015 IEEE 22nd International Conference on High Performance Computing Workshops (HiPCW), pp. 65-65. IEEE, 2015.

[3] G. Dsouza, G. Rodriguez, Y. Al-Nashif, and S. Hariri, “Building Resilient Cloud Services using DDDAS and Moving Target Defense,” International Journal of Cloud Computing, Vol 2, No. 2/3, 2013, pp. 171-190.

[4] H. Kurra, Y. Al-Nashif, and S. Hariri, “Resilient Cloud Data Storage Services,” proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, Article No. 7

[5] G. Dsouza, H. Alipour, S. Hariri, Y. Al-Nashif, and M. Eltoweissy, "Cloud Resilient Architecture," in Proceedings of the 1st IBM Cloud Academy Conference (ICA CON 2012), Research Triangle Park, NC, April 19-20, 2012.

[6] C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, J. Hughes, “Autonomic Resilient Cloud Management (ARCM) Design and Evaluation,” In IEEE 2014 International Conference on Cloud and Autonomic Computing (ICCAC), p. 44-49.

[7] D. Thebeau, B. Reidy, R. Valerdi, A. Gudagi, H. Kurra, Y. Al-Nashif, S. Hariri, and F. Sheldon, “Improving Cyber Resiliency of cloud application services by applying software behavior encryption (SBE)”, 2014 Conference on Systems Engineering Research.

[8] E. Blasch, Y. Al-Nashif, and S. Hariri. “Static Versus Dynamic Data Information Fusion Analysis Using DDDAS for Cyber Security Trust,” Procedia Computer Science 2014;29:1299--1313.

[9] D.L. Pipkin. “Information security: protecting the global enterprise,” Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2000.

[10] M. Roesch, “Snort - Lightweight Intrusion Detection for Networks”, 13th Systems Administration Conference - LISA 1999.

[11] V. Paxson, “Bro: a system for detecting network intruders in real-time.”, Computer Networks (Amsterdam, Netherlands: 1999), 31(23–24):2435–2463, 1999.

[12] K. Scarfone and P. Mell, “Guide to Intrusion Detection and Prevention Systems (IDPS),” Computer Security Resource Center (National Institute of Standards and Technology) (800–94). 2007, Retrieved 1 January 2010.

[13] Y.B. Al-Nashif, A. Kumar, S. Hariri, Y. Luo, F. Szidarovszky, and G. Qu: “Multi-Level Intrusion Detection System (ML-IDS).” ICAC 2008: 131-140, 2008

[14] L. Ertöz, E. Eilertson, A. Lazarevic, P. Tan, V. Kumar, J. Srivastava, and P. Dokas, “Minds - minnesota intrusion detection system”, 2004 Next generation data mining, pp.199-218.

[15] D. E. Denning, “An intrusion-detection model”, IEEE Trans. Softw. Eng., 13(2):222–232, 1987.

[16] H.S. Javitz and A. Valdes, “The nides statistical component: Description and justification”, Technical Report, SRI International Menlo Park, California, 1994.

[17] H. Alipour, Y. Al-Nashif, and S. Hariri, “DNS Anomaly Behavior Analysis against Cyber Attacks”, submitted to ACM Transactions on Internet Technology.

[18] H. Alipour, Y. Al-Nashif, and S. Hariri, “IEEE 802.11 Anomaly Behavior Analysis”, submitted to IEEE Transactions on Information Forensics and Security

[19] R.P. Viswanathan, Y. Al-Nashif, and S. Hariri “Application Attack Detection System (AADS): An Anomaly Based Behavior Analysis Approach”, Accepted in the The 9th ACS/IEEE International Conference On Computer Systems and Applications, 2011.

[20] T.F. Lunt and R. Jagannathan, “A prototype real-time intrusion-detection expert system”, In Proceedings of the IEEE Symposium on Security and Privacy, 1988, pages 18–21, 1988.

[21] D. Anderson, T.F. Lunt, H. Javitz, A. Tamaru, and A. Valdes, “Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (nides)”, Technical Report SRI-CSL-95-06, Computer Sci-ence Laboratory, SRI International, 1995.

[22] P.A. Porras and P.G. Neumann, “Emerald: Event monitoring enabling responses to anomalous live disturbances”, In Proceedings of the National Information Systems Security Conference 1997, pages 353–365, 1997.

[23] S. Staniford, J.A. Hoagland, and J.M. McAlerney, “Practical automated detection of stealthy portscans,” Journal of Computer Security, 10(1-2), pp.105-136.2002.

[24] K. Sequeira and M. Zaki, “Admit: anomaly-based data mining for intrusions,” In KDD ’02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 386–395, New York, NY, USA, 2002. ACM.

[25] N. Ye. “A markov chain model of temporal behavior for anomaly detection,” In Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, 2000, pages 171–174, 2000.

[26] K. Yamanishi, J. Takeuchi, G. J. Williams, and P. Milne, “On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms”, In Knowledge Discovery and Data Mining, pages 320–324, 2000.

[27] N. Ye and Q. Chen, “An anomaly detection technique based on a chi-square statistic for detecting intrusions into in-formation systems”, Quality and Reliability Engineering International 17, no. 2 (2001): 105-112.

[28] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, “A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data,” In Applications of data mining in computer security 2002 (pp. 77-101). Springer US.

[29] C.C. Aggarwal and P.S. Yu, “Outlier detection for high dimensional data,” In SIGMOD Conference, 2001.

[30] M.M. Breunig, H. Kriegel, R.T. Ng, and J. Sander, “LOF: identifying density-based local outliers,” In ACM sigmod record, vol. 29, no. 2, pp. 93-104. ACM, 2000.

[31] E. M. Knorr and R. T. Ng, “Algorithms for mining distance-based outliers in large datasets”, In Proc. 24th Int. Conf. Very Large Data Bases, VLDB, pages 392–403, 24–27 1998.

[32] S. Ramaswamy, R. Rastogi, and K. Shim. “Efficient algorithms for mining outliers from large datasets”. pages 427–438, 2000.

[33] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, “Specification-based anomaly

detection: a new approach for detecting network intrusions," In CCS'02: Proceedings of the 9th ACM conference on Computer and communications security, pages 265–274, New York, NY, USA, 2002.

[34] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection", Information Sciences 177, no. 18 (2007): 3799-3821.

[35] Cloud Security Alliance, "Security as a Service," [Online] Available: https://cloudsecurityalliance.org/research/secaas/, [Accessed January 2013].

[36] M. Rosenblum and T. Garfinkel, "When virtual is harder than real: security challenges in virtual machine based computing environments," in 10th conference on Hot Topics in Operating Systems, Berkeley, 2005.

[37] M. Schmidt, L. Baumgartner, P. Graubner, D. Bock, and B. Freisleben, "Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments," in 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 2011.

[38] D. Goodin, "Webhost Hack Wipes Out Data for 100,000 Sites," [Online]. Available: http://www.theregister.co.uk/2009/06/08/webhost_attack/. [Accessed 15 January 2013].

[39] V.S. Subashini, "A survey on security issues in service delivery models of cloud computing," Journal of Network and Computer Applications, vol. 34, pp. 1-11, 2011.

[40] R. Bhadauria and S. Sanyal, "Survey on Security Issues in Cloud Computing and Associated Mitigation Techniques," International Journal of Computer Applications, vol. 47, no. 18, pp. 47-66, 2012.

[41] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey on security issues and solutions at different layers of Cloud computing," The Journal of Supercomputing, pp. 1-32, 2012.

[42] H. Zeng, "Research on Developing an Attack and Defense Lab Environment for Cross Site Scripting Education in Higher Vocational Colleges," In IEEE 2013 Fifth International Conference on Computational and Information Sciences (ICCIS), (pp. 1971-1974).

[43] M.S. Siddiqui, D. Verma, "Cross site request forgery: A common web application weakness," 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN)

[44] G. Pék, L. Butty´an, and B. Bencsáth, "A survey of security issues in hardware virtualization," ACM Computing Surveys (CSUR) 45, no. 3 (2013): 40.

[45] M. Abbasy and B. Shanmugam, "Enabling Data Hiding for Resource Sharing in Cloud Computing Environments Based on DNA Sequences," in IEEE World Congress, 2011.

[46] J. Feng, Y. Chen, D. Summerville, W. Ku, and Z. Su, "Enhancing cloud storage security against roll-back attacks with a new fair multi-party non-repudiation protocol," in Consumer Communications and Networking Conference, 2011.

[47] L. Kaufman, "Data security in the world of cloud computing," IEEE Security and Privacy Journal, vol. 7, no. 4, pp. 61-64, 2009.

[48] "OWASP Top Ten Project," [Online], Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project [Accessed January 2013].

[49] Q. Nguyen and A. Sood, "Designing SCIT architecture pattern in a Cloud-based environment," in IEEE/IFIP

41st International Conference on Dependable Systems and Networks Workshops, 2011.

[50] P. Zech, "Risk-Based Security Testing in Cloud Computing Environments," in IEEE Fourth International Conference on Software Testing, Verification and Validation, 2011.

[51] [Online]. Available: http://www.cyber.st.dhs.gov/docs/National_Cyber_Leap_Year_Summit_2009_Co-Chairs_Report.pdf. [Accessed January 2013].

[52] S. Forrest, A. Somayaji, and D. H. Ackley, Building diverse computer systems. IEEE, 1997, pp. 67–72.

[53] J. P. G. Sterbenz and P. Kulkarni, "Diverse Infrastructure and Architecture for Datacenter and Cloud Resilience," 2013 22nd International Conference on Computer Communication and Networks (ICCCN 2013), pp. 1–7, 2013.

[54] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: a moving target IPv6 defense," in 2011 Military Communications Conference (MILCOM), Maryland, 2011.

[55] R. Zhuang, S. Zhang, S. A. DeLoach, X. Ou, and A. Singhal, "Simulation-based approaches to studying effectiveness of moving-target network defense," in National Symposium on Moving Target Research, Annapolis, 2012.

[56] S. Narain, "Moving Target Defense With Configuration Space Randomization," [Online], Available: https://www.ncsi.com/nsatc11/presentations/thursday/emerging_technologies/narain.pdf [Accessed 30 January 2013].

[57] K. Kim, "ROAFTS: A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support," in IEEE CS 1998 High-Assurance Systems Engineering (HASE) Symp, Washington, D.C., 1998.

[58] A. Tyrrell, "Recovery Blocks and Algorithm Based Fault tolerance," in 22nd EUROMICRO Conference, 1996.

[59] K. Kim and H. Welch, "Distributed Execution of Recovery Blocks:An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications," IEEE transactions on Computers, vol. 38, no. 5, pp. pp. 626-636, 1989.

[60] W. Toy, "Fault-Tolerant Computing," in Advances in Computers, Academic Press, 1987, pp. 201-279.

[61] A. Avizienis, "The N-Version Approach to fault Tolerant Software," IEEE transactions on Software Engineering, Vols. SE-11, no. 12, 1985.

[62] D. Evans, A. Nguyen-Tuong and J. Knight, "Effectiveness of Moving Target Defenses," in Advances in Information Security, Springer, 2011, pp. 29-39.

[63] "PaX Homepage," [Online], Available: http://pax.grsecurity.net/ [Accessed October 2012].

[64] E. Barrantes, D. Ackley, S. Forrest, T. Palmer, D. Stefanovic, and D. Zovi, "Intrusion Detection: Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," in 10th ACM Conference on Computer and Communications Security, 2003.

[65] C. Cadar, P. Akritidis, M. Costa, J.P. Martin, and M. Castro, "Data Randomization," Microsoft Research, 2008.

[66] P. Verissimo, A. Bessani, and M. Pasin, “The TClouds Architecture: Open and Resilient Cloud-of-clouds Computing,” in IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops, 2012.

[67] G. Vallee, C. Engelmann, A. Tikotekar, T. Naughton, K. Charoenpornwattana, C. Leangsuksun and S. Scott, “A Framework for Proactive Fault Tolerance,” in Third International Conference on Availability, Reliability and Security, 2008.

[68] A. Keromytis, G. R., S. Sethumadhavan, S. Stolfo, Y. Junfeng, A. Benameur, M. Dacier, M. Elder, D. Kienzle and A. Stavrou, “The MEERKATS Cloud Security Architecture,” in 32nd International Conference on Distributed Computing Systems Workshops, 2012.

[69] D. Luo and J. Wang, “CC-VIT: Virtualization Intrusion Tolerance Based on Cloud Computing,” in 2nd International Conference on Information Engineering and Computer Science, 2010.

[70] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, “Enabling secure VM-vTPM migration in private clouds,” ACSAC '11, pp. 187–196, Dec. 2011.

[71] US Air Force Chief Scientist, “Report on Technology Horizons: A Vision for Air Force Science and Technology During 2010–2030,” Volume 1, Technical Report AF/ST-TR-10-01-PR, Department of the Air Force, Washington, DC, 2010.

[72] [Online], Available: http://www.nitrd.gov/pubs/CSIA_IWG_%20Cybersecurity_%20GameChange_RD_%20Recommendations_20100513.pdf [Accessed 15 January 2013].

[73] G. Rodríguez, M. Martín, P. González, J. Touriño and R. Doallo, “CPPC: A compiler-assisted tool for portable checkpointing of message-passing applications,” Concurrency and Computation: Practice & Experience, vol. 22, no. 6, pp. 749-766, 2010.

[74] B.U. Kim, Y. Al-Nashif, S. Fayssal, S. Hariri, and M. Yousif, “Anomaly-based fault detection in pervasive computing system.” In Proceedings of the 5th international Conference on Pervasive Services (Sorrento, Italy, July 06 - 10, 2008). ICPS '08. ACM, New York, NY, pp. 147-156.

[75] [Online] http://nvd.nist.gov/cvss.cfm , [Accessed January 2014]

[76] S. Latif-Shabgahi, “An Integrated Voting Algorithm for Fault Tolerant Systems,” in Proc. International Conference on Software and Computer Applications (IPCSIT), vol. 9, 2011.

[77] [Online] Available: http://code.google.com/p/mrs-mapreduce/ [Accessed May 2014]

[78] “The CPPC Project: Controller/comPiler for Portable Checkpointing,” [Online] Available: http://cppc.des.udc.es/ [Accessed July 2016]

[79] [Online] http://www.hdfgroup.org/HDF5/ [Accessed in June 2012]

[80] G. Rodríguez, M.J. Martín, P. González, and J. Touriño, “A heuristic approach for the automatic insertion of checkpoints in message-passing codes”, Journal of Universal Computer Science, Vol. 15, No. 14, pp.2894–2911.

[81] [Online] Available: https://www.openstack.org/ [Accessed May 2014]

[82] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao, “Autonomia: an autonomic computing environment,” In 2003 IEEE International Conference of Performance, Computing, and Communications

Conference, 2003 (pp. 61-68).

[83] Cox, Don P., Youssif Al-Nashif, and Salim Hariri. “Application of autonomic agents for global information grid management and security.” In Proceedings of the 2007 summer computer simulation conference, pp. 1147-1154. Society for Computer Simulation International, 2007.

[84] Chen, Huoping, Youssif B. Al-Nashif, Guangzhi Qu, and Salim Hariri. “Self-configuration of network security.” In Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, pp. 97-97. IEEE, 2007.

[85] J. D. and S. G., “MapReduce: Simplified Data Processing on Large Clusters,” in Sixth Symposium on Operating Systems Design and Implementation, 2008.

[86] “Apache Hadoop,” [Online] Available: http://hadoop.apache.org/ [Accessed July 2016]

[87] “Oracle Virtualbox,” [Online] Available: https://www.virtualbox.org/ [Accessed July 2016]

[88] [Online] Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html [Accessed July 2016]

[89] [Online] Available: http://college.cengage.com/mathematics/larson/elementary_linear/5e/students/ch08-10/chap_10_2.pdf

[90] “IBM HS22 Blade Server” [Online] Available: http://www-03.ibm.com/systems/bladecenter/index.html

[91] “VMware vSphere,” [Online] Available: http://www.vmware.com/products/vsphere/mid-size-and-enterprise-business/overview.html [Accessed on May 2011].

[92] “Free Mersenne Prime Search Software, Prime95 Version 28.9,” [Online] Available: http://www.mersenne.org/download/

[93] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, “Mibench: A free,commercially representative embedded benchmark suite”. In WWC ’01: Proceedings of the Workload Characterization, pp. 3–14, Washington, DC,USA.

[94] [Online] Available: http://www.emc.com/about/news/press/2011/20110628-01.htm

[95] M. Robertson, “300+ Hours of Video Uploaded to YouTube Every Minute” [Online] Available: http://www.reelseo.com/youtube-300-hours/

[96] [Online] “Subscriptions to Cloud Storage Services to Reach Half-Billion Level This Year”, Available: https://technology.ihs.com/410084/subscriptions-to-cloud-storage-services-to-reach-half-billion-level-this-year

[97] [Online] Cisco Visual Networking Index Report, Available: http://blogs.cisco.com/sp/ip-traffic-to-quadruple-by-2015/#utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+CiscoBlogSp360ServiceProvider+%28Cisco+Blog+%C2%BB+SP360%3A+Service+Provider%29.

[98] [Online] Available: www.futurecloudcomputing.net

[99] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A View of Cloud Computing.” Comm. ACM, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[100] Y. Tang, P.P. Lee, J. Lui, and R. Perlman, “Secure Overlay Cloud Storage with Access Control and Assured Deletion”, IEEE transactions on dependable and secure computing, vol. 9, no. 6, November/December 2012.

[101] [Online] Amazon, “case studies,” Available: http://aws.amazon.com/solutions/case-studies/#backup, 2012.

[102] [Online] Available:www.futurecloudcomputing.net.

[103] [Online] Available:http://www.techvibes.com/blog/canadians-are-cautious-and-conservative-when-it-comes-to-cloud-computing-2011-06-07.

[104] [Online] Study Conducted in March by AFCOM - Available:http://www.afcom.com/communique/communique_1.html.

[105] [Online] Research by Ponemon Institute, Benchmark study of 41 US datacenters, Available: http://www.emersonnetworkpower.com/en-US/Brands/Liebert/Documents/White%20Papers/sl-24659.pdf.

[106] S. Kamara, and K. Lauter, “Cryptographic cloud storage.” In International Conference on Financial Cryptography and Data Security, pp. 136-149. Springer Berlin Heidelberg, 2010.

[107] “cvechecker,” [Online] Available: https://github.com/sjvermeu/cvechecker/wiki [Accessed July 2016]

[108] [Online] Available: http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[109] B. Kaliski, “A survey of encryption standards,” IEEE micro, 13(6), pp.74-81, 1993.

[110] “Frequency Hopping Systems,” [Online] Available: http://www.ti.com/lit/an/swra077/swra077.pdf [Accessed July 2016]

[111] “Certificates for SSL Applications,” [Online] Available: http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04s02.html#cert3-fig [Accessed July 2016]

[112] “OpenSSL DES APIs,” [Online] Available: http://blog.fpmurphy.com/2010/04/openssl-des-api.html [Accessed July 2016]

[113] [Online] Available: http://my.opera.com/securitygroup/blog/2009/09/29/512-bit-rsa-key-breaking-developments [Accessed July 2013]

[114] A. Thiruneelakandan, and T. Thirumurugan, “An approach towards improved cyber security by hardware acceleration of OpenSSL cryptographic functions," In Electronics, Communication and Computing Technologies (ICECCT), 2011 International Conference on, pp. 13-16. IEEE, 2011.

[115] J. Black, M. Cochran, T. Highland, “A Study of the MD5 Attacks: Insights and Improvements,” In International Workshop on Fast Software Encryption, pp. 262-277. Springer Berlin Heidelberg, 2006.

[116] Schwartz, Mathew J. “Dropbox Accused of Misleading Customers on Security.” Information Week. May 16, 2011. http://www.informationweek.com/news/storage/security/229500683 [Accessed July 2016]

[117] P.K. Manadhata, and J.M. Wing, “An attack surface metric,” IEEE Transactions on Software Engineering 37, no. 3 (2011): 371-386.

[118] “Attack Surface Analyzer,” [Online] Available: http://www.microsoft.com/en-us/download/details.aspx?id=24487 [Accessed July 2016]

[119] “Flawfinder,” [Online] Available: http://www.dwheeler.com/flawfinder/ [Accessed July 2016]

[120] “Nessus Vulnerability Scanner,” [Online] Available: http://www.tenable.com/products/nessus [Accessed July 2016]

[121] “Enterprise Vulnerability Management Software for Dynamic IT Environments”, [Online] Available: http://www.beyondtrust.com/Products/RetinaCSThreatManagementConsole/ [Accessed July 2016]

AFOSR Deliverables Submission Survey

Response ID:6619 Data

1.

1. Report Type

Final Report

Primary Contact E-mail
Contact email if there is a problem with the report.

hariri@email.arizona.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

520-621-4378

Organization / Institution name

The University of Arizona

Grant/Contract Title
The full title of the funded effort.

DDDAS-based Resilient Cyberspace (DRCS)

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-12-1-0241

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Salim Hariri

Program Manager
The AFOSR Program Manager currently assigned to the award

Dr. Frederica Darema

Reporting Period Start Date

05/01/2012

Reporting Period End Date

04/30/2016

Abstract

Dynamic Data-Driven Application Systems (DDDAS) applications will be widely deployed to optimize the operations of cyber infrastructures and mission critical applications. Consequently, it is critically important for DDDAS environments to operate resiliently against any type of cyberattacks (either known or unknown). In this project, we focused on the development of resilient algorithms, middleware, and DDDAS-based applications that can continue operating normally in spite of the occurrence of cyberattacks, faults or accidents that could be triggered by malicious or natural events.

The main contributions of this research are the followings:

- A methodology to build resilient DDDAS (rDDDAS) environment: Utilizing Moving Target Defense (MTD) and Software Behavior Encryption (SBE) techniques to make it extremely difficult for hackers to exploit existing vulnerabilities or compromise DDDAS environments. We show that with the attributes of redundancy, diversity, and shuffling, it is possible to meet any security and resilience requirements at runtime.

- Resilient Cloud Services (RCS) and Middleware: By using cloud services and virtual machines, we created diversified environments and also adopted portable checking pointing technique and showed acceptable performance and low overhead on several general applications.

- Analytical Resilient Modeling: We provide a method for analyzing the system resiliency to quantify the probability of successful attacks when we use Software Behavior Encryption (SBE). We leverage Common Vulnerabilities and Exposures (CVE) to estimate the probability of successful attack and quantify the resilience of the SBE algorithm against the existing vulnerabilities. We showed that the probability of a successful attack can be reduced to almost zero if we can use three or more diverse versions in the SBE algorithm.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

SF298.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

rDDDAS-Final-Report-final.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

1. J. Pacheco, C. Tunc, and S. Hariri, "Design and Evaluation of Resilient Cyber Infrastructures for Smart Cities," to be presented at the IEEE Second International Smart Cities Conference (ISC2 2016), September 2016, Trento, Italy.
2. G. Dsouza, G. Rodriguez, Y. Al-Nashif, and S. Hariri, "Building Resilient Cloud Services using DDDAS and Moving Target Defense," International Journal of Cloud Computing, Vol 2, No. 2/3, 2013, pp. 171-190.
3. G. Dsouza, S. Hariri, Y. Al-Nashif, and G. Rodriguez, " Resilient Dynamic Data Driven Application Systems (rDDDAS)", Proceedings of International Conference on Computational Science, Barcelona, Spain, 5-7 June, 2013.
4. G. Dsouza, H. Alipour, S. Hariri, Y. Al-Nashif, and M. Eltoweissy, "Cloud Resilient Architecture," in Proceedings of the 1st IBM Cloud Academy Conference (ICA CON 2012), Research Triangle Park, NC, April 19-20, 2012.
5. C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, J. Hughes, "Autonomic Resilient Cloud Management (ARCM) Design and Evaluation," in 2014 IEEE International Conference on Cloud and Autonomic Computing (ICCAC), 2014 p. 44-49.
6. C. Tunc, S. Hariri, and A. Battou, "A Design Methodology for Developing Resilient Cloud Services (RCS)," Handbook of System Safety and Security: Cyber Risk and Management, Cyber Security, Threat Analysis, Functional Safety, Software Systems, and Cyber Physical Systems. Edited by Edward Griffor, Elsevier Inc., to be published in 2016.
7. E. Blasch, Y. Al-Nashif, S. Hariri. "Static Versus Dynamic Data Information Fusion Analysis Using DDDAS for Cyber Security Trust. Procedia Computer Science 2014; 29:1299-1313.
8. E. Blasch, Y. Badr, S. Hariri, and Y. Al-Nashif, "Fusion Trust Service Assessment for Crisis Management Environments," pages 389-420, a chapter in a book "Fusion Methodologies in Crisis Management: Higher Level Fusion and Decision Making," Editors: Galina Rogova, Peter Scott, Springer, ISBN 978-3-319-22527-2, (2016).
9. Y. Badr, S. Hariri, Y. Al-Nashif, E. Blasch, "Resilient and Trustworthy Dynamic Data-Driven Application

Systems (DDDAS) Services for Crisis Management Environments," Procedia Computer Science, ICCS 2015 International Conference On Computational Science, Volume 51, 2015, Pages 1–15, 2015.

2. New discoveries, inventions, or patent disclosures:

Do you have any discoveries, inventions, or patent disclosures to report for this period?

No

Please describe and include any notable dates

Do you plan to pursue a claim for personal or organizational intellectual property?

Changes in research objectives (if any):

Change in AFOSR Program Manager, if any:

Extensions granted or milestones slipped, if any:

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Jul 31, 2016 18:03:40 Success: Email Sent to: hariri@email.arizona.edu